Facultade de Informática
UNIVERSIDADE DA CORUÑA

TRABALLO FIN DE GRAO
GRAO EN ENXEÑARÍA INFORMÁTICA
MENCIÓN EN TECNOLOXÍAS DA INFORMACIÓN

Gestor de noticias y vídeos corporativos

Estudante: Marcos Puerta Caramés
Dirección: Mariano Javier Cabrero Canosa

A Coruña, xuño de 2020.

No importa cuán oscura sea la noche.
Al final, siempre se acabará haciendo de día.
Brook

Agradecimientos

Primero de nada a mi familia, por el apoyo incondicional tanto emocional como económico durante todo este proceso de aprendizaje.

A mi gatetes, los que están y los que ya no están por hacerme compañía.

A mis amigos de la facultad, por ayudarme, entretenerme y hacerme disfrutar. Mención especial a Isa y Alicia por ayudarme con este proyecto si lo necesitaba y a André y Dani siempre al pie del cañón.

A mis amigos de siempre, que siempre están ahí, preguntando quien baja por la tarde o apuntadose a cualquier plan.

A Silvia, y su conexión 24/7 con mi persona, por su atención, ánimos y apoyo.

A los profesores, por transmitirnos sus conocimientos.

A mi tutor Mariano, por estar presente y dejarme la libertad que necesitase.

Y a todas esas personas que no se hayan podido dar por aludidas pero saben que están ahí y se merecen aparecer.

Resumen

Internet es uno de los medios más potentes para hacer publicidad a día de hoy. La posibilidad de disponer de un medio de difusión de vídeos y noticias propio, es una buena manera para darse a conocer.

En este proyecto se presenta una plataforma web *responsive* de gestión de autopublicidad basada en la difusión de noticias y en la publicación de vídeos de la marca es lo que se llevara a cabo. Constará de un sistema de gestión diseñado para ser útil y sencillo, con el cual se podrán añadir y modificar dichos vídeos y noticias. Las noticias podrán ser escritas en *HTML* para una perfecta personalización y los vídeos podrán ser importados de otras plataformas o subir los suyos propios.

La interfaz del cliente será amigable y sencilla, de forma que la pantalla principal presente un vídeo acompañado de una lista de reproducción al estilo *Youtube* y un carrusel de las últimas noticias tal y como se ven en los informativos, pudiendo acceder a ellas con un simple *click*. Además disponen de una sección solo con noticias y una nube de etiquetas con las más utilizadas.

Abstract

Internet is one of the most powerful means of advertising today. The posibility of having your own news and video broadcasting service is a good way to make yourself known.

A *responsive* web platform for self-advertising management based on the broadcasting news and brand videos is what will do. It will have a management system designed to be useful and simple, with which these videos and news can be added and modified. The news can be written in *HTML* for a perfect customization and the videos can be imported from other platforms or upload your own.

The client interface will be friendly and simple, so with a simple glance you will have a main video along with a playlist like *Youtube* and a carousel of the latest news as seen in the news bulletins , being able to access them with a simple *click*. They also have a section only with news and a tag cloud with the most used.

Palabras clave:

- Java
- React
- Aplicación web
- Videos y noticias
- MySql
- Npm
- Spring
- JavaScript
- CSS
- Hibernate
- JPA
- Eclipse
- Visual Studio Code
- GIT
- Maven
- PostMan
- Redux
- Bootstrap
- ...

Keywords:

- Java
- React
- Web App
- Videos and news
- MySql
- Npm
- Spring
- JavaScript
- CSS
- Hibernate
- JPA
- Eclipse
- Visual Studio Code
- GIT
- Maven
- PostMan
- Redux
- Bootstrap
- ...

Hardware y software utilizado

Para el desarrollo de la plataforma he utilizado un ordenador montado a piezas con las siguientes especificaciones:

- Intel i5 9600K 4.6Ghz Hexa Core11
- MSI GeForce GTX 1080Ti GAMING 11Gb GDDR5X18
- MSI MAG Z390 TOMAHAWK18
- 24 GB RAM (2x3000MHz CL15, 1x2666Mhz CL16)

Como software, a grandes rasgos, se ha utilizado **Eclipse** para elaborar el backend y **Visual Studio Code** para elaborar la interfaz de usuario.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introdución

En esta introducción el objetivo es describir las motivaciones, objetivos del proyecto y la organización de la memoria.

1.1 Motivación

La motivación principal de este **Proyecto Fin de Grado** han sido las ganas de hacer una plataforma web con características variadas utilizando tanto recursos aprendidos en asignaturas cursadas, así como, recursos y tecnologías que no había utilizado nunca pero que mezcladas con las ya aprendidas eran las suficientes para crear dicha plataforma.

También existe otra motivación. Todos hemos visto esas páginas de empresas, que son tan genéricas y repetitivas, sin casi interacción, o con una interacción que no permite al usuario informarse realmente de dicha empresa. Lo que se intenta con esta plataforma es que esto no sea así.

Por último hay una motivación personal de la creación de una plataforma con la poca experiencia obtenida para montar una plataforma completo y de trabajar con tecnologías punteras, que a su vez son diferentes de las que había aprendido durante la carrera.

1.2 Objetivos

El objetivo principal es la construcción de una plataforma *responsive* que autopublicite al cliente. Ésta contendrá tanto vídeos como noticias de la empresa, de manera que el usuario que acceda a ella la encuentre atractiva y útil.

La plataforma debe ser fácil de utilizar por parte del administrador y debe constar con un panel para añadir y modificar tanto los vídeos como las noticias. También debe permitir modificar la visibilidad.

El usuario podrá elegir el vídeo que desea ver y tendrá a primera vista un carrusel de

noticias en constante movimiento. Además tendrá a su disposición una sección dedicada solo a las noticias, donde podrá navegar por ellas.

Como añadido, las noticias dispondrán de etiquetas. Mediante éstas, se podrá acceder a todas las noticias que contengan dicha etiqueta.

En conclusión, una plataforma *responsive*, atractiva y sencilla de usar tanto para el cliente como para el usuario que la visita.

1.3 Organización

A continuación se expondrán una pequeña explicación de cada apartado de este documento sobre el **Trabajo Fin de Grado** que se ha realizado. Para hacer el documento se ha utilizado *LaTeX* con la plantilla ofrecida por la facultad.

1. **Introducción:** breve descripción del proyecto donde se busca que el lector pueda responderse a si mismo qué ha motivado la realización de este proyecto y qué objetivos tiene. Finaliza con la enumeración de contenidos de la memoria del proyecto.

2. **Estado del arte y conceptos previos:** se expone cómo esta actualmente el mundo del diseño web para empresas noveles a grandes rasgos y el porqué de llevar a cabo este proyecto.

3. **Fundamentos tecnológicos:** se citan las tecnologías utilizadas y cuando es preciso se indica el porqué de su utilización.

4. **Metodología:** se explica cuál es la metodología que se utiliza, qué modificaciones o aspectos resultan más interesantes y porqué.

5. **Requisitos del sistema:** se desarrolla una lista de cómo ha de funcionar la aplicación y qué necesita para ello.

6. **Diseño de la aplicación:** se explica y muestra cómo ha sido diseñada la aplicación y su funcionamiento.

7. **Pruebas:** se muestran las distintas pruebas que se han realizado para buscar fallos y comprobar el correcto funcionamiento de la aplicación.

8. **Planificación:** se muestra cuál ha sido el resultado de la planificación llevada a cabo y las conclusiones que se han sacado.

9. **Conclusiones:** se explican cuáles son las conclusiones de haber realizado el proyecto y se analiza lo aprendido.

Capítulo 2

Estado del arte y conceptos previos

A día de hoy para hacer una página web de una empresa tenemos dos opciones:

- Utilizar herramientas estilo *1&1* para realizar una web genérica.
- Encargar que una empresa te haga y gestione tu web exclusivamente para tí.

En este apartado analizaremos ambas posibilidades.

2.1 Herramientas genéricas

La herramienta más conocida para hacer páginas web para empresas es la propia ***IONOS by 1&1***.

Una de las opciones de las que disponen las empresas de *hosting*, y que a menudo anuncian, es el diseño de webs a partir de unos elementos que vas colocando a tu gusto. Uno de los ejemplos habituales son *scrolls* con todas opciones en una misma página o divida, sin embargo, suelen ser contenidos poco variables, con alguna imagen o vídeo fijo que se centran más en el diseño y en quedar bonito que en la propia información.

El principal problema que encontramos en estas soluciones es que pese a que este ”todo hecho” tener una web de estas características obliga a usar su propio dominio. Por una parte ofrecen gran cantidad de servicios, de hecho están dentro de sus ofertas para atraer al público. Sin embargo, recientemente han comenzado a tener quejas sobre sus servidores y sobre su estabilidad.

2.2 Encargo

Hacer una web de encargo es la mejor opción si se tiene claro lo que se quiere mostrar de la empresa. Sin embargo, este tipo de trabajos los puede hacer mucha gente con muy diferentes

skills, por tanto la seguridad de que todo vaya a salir bien depende mucho de la suerte o de los conocimientos del desarrollador.

El principal problema de esta opción suele ser la capacidad para modificar tu propia página, o la falta de una idea previa sobre lo que se te ofrecer.

2.3 Conclusión

Por una parte, estar atado a una empresa que se encargue de absolutamente todo conlleva los inconvenientes de la falta de opciones. Por otra, encargar a un tercero el diseño, obliga a tener unas ideas muy claras o pagar un extra por cada cambio. Además el diseño no lo es todo, también es importante tener un buen sistema para poder añadir vida a la web.

La solución que se expone en este **Trabajo Fin De Grado** es una mezcla de ambas opciones, y que a su vez, elimina las mayores restricciones. Una plataforma que deje subir al administrador tanto vídeos como noticias de su empresa, así como, total libertad de *hosting* o de opciones a mayores con otros servicios. Todo será totalmente independiente pero al mismo tiempo se tendrá la fiabilidad de poder ir actualizando su plataforma sin la necesidad de tener una persona al cargo de ello.

Capítulo 3

Fundamentos Tecnológicos

En este capítulo se analizará el software que se ha utilizado en el proyecto, para qué se ha utilizado y en las ocasiones en que haya más opciones viables, el porqué de la elección.

3.1 Lenguajes

3.1.1 Java

El lenguaje seleccionado para realizar la capa *Modelo* y la capa *Servicios*, así como los controladores ha sido **Java**. Este nos permite la integración de todas la tecnologías del *backend* con mucha facilidad. He escogido este lenguaje por la experiencia previa que tenía con el, además de la simpleza a la hora de hacer un *backend* sólido.

3.1.2 SQL

SQL es un lenguaje de dominio específico utilizado en programación, diseñado para administrar y recuperar información de sistemas de bases de datos relacionales [1]. Debido a ser el estándar, éste ha sido el lenguaje utilizado para la creación de nuestra base de datos.

3.1.3 JavaScript

JavaScript es el lenguaje por excelencia para realizar la aplicación del lado del lado cliente en aplicaciones web, ya que aporta gran variedad de librerías y gran cantidad de funciones. Además todos los navegadores modernos interpretan el código **JavaScript** integrado en las páginas web.

Se ha escogido este lenguaje básicamente por su integración con una gran cantidad de librerías entre la que destaca **React**, que es la base de nuestra capa cliente.

Pese a no haber trabajado mucho con este lenguaje anteriormente sus grandes posibilidades hace que merezca la pena adentrarse en él.

3.1.4 HTML

HTML, siglas en inglés de **HyperText Markup Language**, es el lenguaje de marcado para la elaboración de las páginas web.

3.1.5 CSS

CSS, también conocido como **Cascading Stylesheets**, es el lenguaje utilizado para el diseño gráfico de un documento **HTML**. Con el podemos definir los estándares y los diseños de la plataforma desde un solo documento.

3.2 Herramientas

3.2.1 Eclipse

Para la realización del *backend* el *IDE* utilizado ha sido **Eclipse**.

Se ha seleccionado esta plataforma debido a la experiencia previa con ella a la hora de realizar capas modelo y capas cliente en otros proyectos. Además disponíamos de una preconfiguración en la cual estaban endebidas las principales tecnologías presentes en el *backend*.

3.2.2 Visual Studio Code

Visual Studio Code es un editor de código fuente desarrolado por *Microsoft* para *Windows*, *Linux* y *macOS*. Esta herramienta multiplataforma nos brinda un diseño mucho más amigable a la vista del desarrollador que otros *IDEs* del mercado.

Se ha utilizado este *IDE* para la realización la la aplicación cliente por su gran integración con **JavaScript**, **HTML** y **CSS** permitiendo una visualización del código limpia y fácil de leer.

Además dispone de un lector **SQL** que nos ha servido para la creación del código que genera la base de datos.

3.2.3 MySQL WorkBench

Se ha utilizado la herramienta de **MySQL WorkBench** para el manejo en tiempo real de la base de datos. El uso de esta herramienta está motivado por la utilización como base de datos de **MySql**, la cual, es idónea por su facilidad para conectarse con **Java**.

Como extra, nos permite sacar una instantánea del modelo E-R de nuestra base de datos, lo que ha ayudado a realizar las consultas para algunas de las operaciones.

3.2.4 Maven

Maven es una herramienta de software de gestión y construcción de proyectos Java. Con esta herramienta hemos podido manejar el modelo y la capa servicios sin ningún tipo de problema.

Gracias a su **POM** (Project Object Model) se puede describir el proyecto de software a construir, sus dependencias de otros módulos y componentes externos y también el orden de construcción de los elementos.

3.2.5 Herramientas para pruebas

Junit

JUnit es un conjunto de bibliotecas utilizadas en programación para hacer pruebas unitarias de aplicaciones **Java**. Éste viene integrado en **Eclipse** por tanto es el que se ha utilizado en las pruebas de la capa modelo al principio del desarrollo.

PostMan

PostMan es una herramienta de que se utiliza, mayoritariamente, para el testing de *API REST*, aunque también tiene otra funcionalidades. En nuestro caso se ha utilizado para hacer las pruebas de la capa *Servicios* del proyecto. Con esta herramienta se pueden enviar peticiones de manera muy intuitiva y sencilla. Además cuenta con una interfaz muy comoda para llevar a cabo diversas pruebas de manera independiente.

3.2.6 Git

Git es el software de control de versiones más utilizado. Está pensado para que funcione de manera eficiente y confiable hasta cuando se dispone de un grán numero de archivos de código fuente.

Con esta herramienta, hemos utilizado los repositorios de la facultad para ir guardando el progreso y haciendo las diferentes ramas a la hora del desarrollo de nuevas funciones.

3.2.7 Ejecución

Spring Boot

Spring Boot es una de las tecnologías del mundo de Spring que sirve para lanzar nuestras aplicaciones desarrolladas con el *framework* **Spring**. Unifica el proceso de **Maven**, la creación de la aplicación y el despliegue del servidor. Además nos facilita el uso de **Spring** y **JPA/Hibernate**.

NPM

NPM es el sistema de gestión de paquetes por defecto de **Node.Js**, que es el entorno en el cual trabajamos. Desde este gestor tenemos una gran variedad de herramientas a nuestra disposición, de las cuales hacemos uso en el proyecto. Algunos ejemplos pueden ser el *Scroll de noticias* o el *reproductor de vídeos*.

3.3 Frameworks y Librerías

3.3.1 Spring

Spring es el framework en el cual basamos todo el backend. Este framework nos ayuda a diversidad de tareas realizadas, entre las que destacan:

- Contenedor de inversión de control
- Acceso a datos
- Gestión de transacciones
- Gestión del *API REST*
- Seguridad
- ...

3.3.2 Hibernate

Hibernate es una herramienta de mapeo objeto-relacional para la plataforma **Java**. Con ella se facilita el mapeo de atributos entre una base de datos relacional y el modelo de objetos de una aplicación.

3.3.3 React

React es la biblioteca **JavaScript** en la cual se basa todo el *front-end* del proyecto. Ésta es de código abierto y está diseñada para crear interfaces de usuario con el objetivo de facilitar el desarrollo de aplicaciones en una sola página. Esta biblioteca está en auge, así que fue la escogida para llevar a cabo esta parte. Cuenta con una gran biblioteca de recursos que se pueden instalar mediante **NPM** lo cual ha hecho que la experiencia haya sido agradable a la hora de encontrar un problema. Como extra tiene una gran comunidad, por lo que, pese a no haberla utilizado antes, ha sido llevadero de aprender.

3.3.4 Bootstrap

Bootstrap es una de las bibliotecas multiplataforma de código abierto para diseño de sitios aplicaciones web más conocida. Contiene plantilla de diseño para casi cualquier elemento basado en **HTML** o **CSS**, así como extensiones JavaScript. Esta herramienta solo se ocupa del *front-end*, pero es justo lo que buscamos, ya que el proyecto se divide claramente en un *backend* realizado en **Eclipse** y un *front-end* realizado en **Visual Studio Code**. Además nos proporciona grán facilidad a la hora de hacer un diseño *responsive* y que siga los *estandares*.

3.4 Protocolos

3.4.1 HTTP y HTTPS

Ambos son los protocolos de comunicación que permiten transferencias de información en la web. Siendo **HTTPS** la versión segura de **HTTP**.

Capítulo 4
Metodología

Por definición, las **metodologías ágiles** son aquellas que permiten adaptar la forma de trabajo a las condiciones del proyecto, consiguiendo flexibilidad e inmediatez en la respuesta para amoldar el proyecto y su desarrollo a las circunstancias específicas del entorno[2]. Cada proyecto se 'trocea' en pequeñas partes que tienen que completarse y entregarse en pocas semanas. El objetivo es desarrollar productos y servicios de calidad que respondan a las necesidades de unos clientes cuyas prioridades cambian a una velocidad cada vez mayor[3].

Alguna de las ventajas de este tipo de metodologías son:

- **Mejora la calidad:** Minimiza los errores en los entregables y mejora la experiencia y las funcionalidad para el cliente.
- **Mayor compromiso:** Mejora la satisfacción del empleado y genera conciencia de equipo.
- **Rapidez:** Acorta los ciclos de producción y minimiza los tiempos de reacción y toma de decisiones.
- **Aumento de la productividad:** Al asignar mejor los recursos, y de forma más dinámica, mejora la producción según las prioridades que tenga la empresa, o el proyecto en nuestro caso.

Los principios y valores en los que se basan las metodologías ágiles tienen como principal característica realizar entregas rápidas y continuas de software funcionando. Poniendo un ejemplo, en el marco de trabajo *scrum*, el proyecto se divide en pequeñas partes que tienen que completarse y entregarse en plazos cortos, llamados *'sprints'*. De esta manera, si hay que realizar cualquier modificación, sólo se hacen cambios en la parte implicada y en muy poco tiempo[3].

Este tipo de metodologías están pensadas para equipos, aunque se pueden adaptar. También están pensadas para desarrollarse en un mes aproximadamente, pero al disponer de más tiempo, solo de una persona, y tener unos objetivos definidos desde el principio se ha podido amoldar perfectamente. Lo que más llama la atención de estas metodologías es la satisfacción de acabar un periodo y tener ya una aplicación funcional, aunque solo pueda realizar una acción. Ello hace que el desarrollador, vea que ya ha realizado un trabajo útil.

La metodología concreta que más ha llamado la atención es la metodología **Scrum** (figura 4.1). Esta metodología utiliza un tipo de incrementos llamados **Sprint**. Un **Sprint** es un intervalo de tiempo de máximo un mes donde se desarrolla aun incremento de un producto potencialmente entregable. Los podríamos considerar mini-proyectos.

Se hablará de cuales son las interacciones en el capítulo 8.

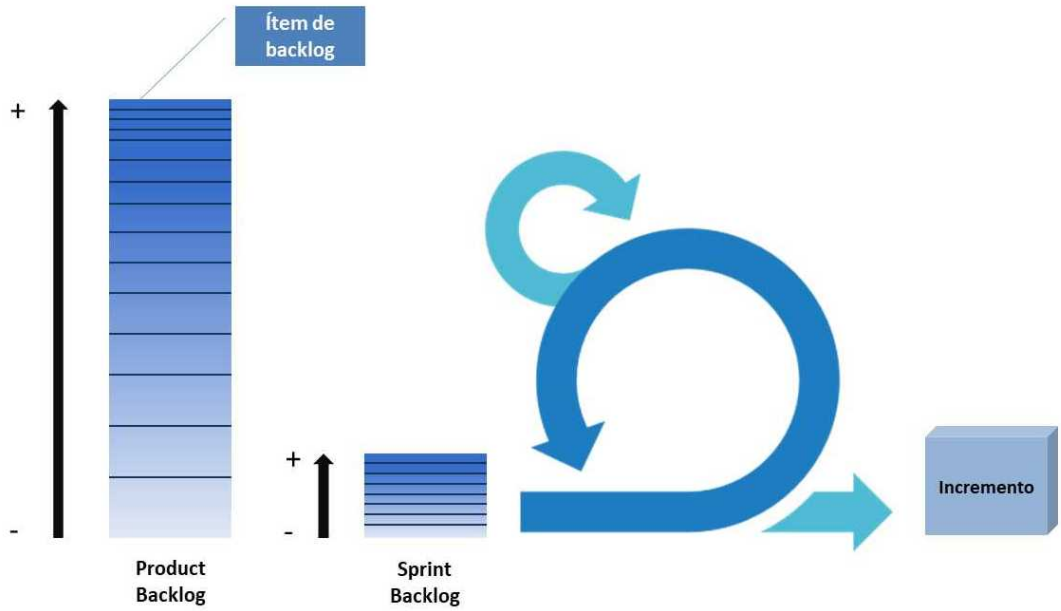

Figura 4.1: Diagrama metodología Scrum

Capítulo 5

Requisitos del sistema

A continuación se muestra una descripción completa del comportamiento del sistema que se ha desarrollado. En ella se incluyen los requisitos funcionales, también conocidos como casos de uso, y los requisitos no funcionales.

5.1 Usuario-Credenciales

Se realizará una descripción de los casos de uso correspondientes al manejo de credenciales por parte de un usuario administrador del sistema.

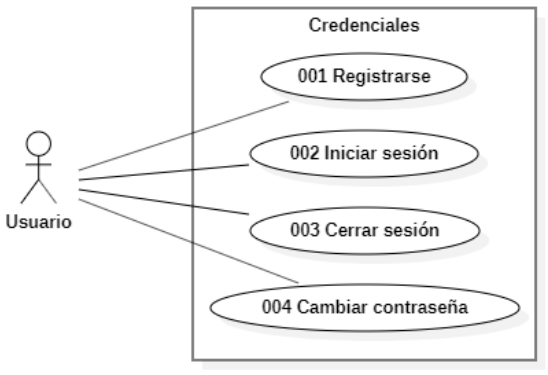

Figura 5.1: Diagrama Usuario-Credenciales

CU-001: Registrarse.		
Descripción	Rellena los campos usuario, contraseña y la confirmación de esta.	
Actores	Usuario.	
Precondiciones	Dispone de los permisos para registrarse.	
Flujo normal	Paso	Acción
	1.	El usuario pulsa en *Iniciar Sesión* en la página principal.
	2.	El usuario pulsa en *Registrarse* en la pantalla de inicio de sesión.
	3.	El usuario introduce un nombre de usuario y contraseña, confirmando esta una segunda vez.
	4.	El usuario pulsa en *Registrarse* y es devuelto a la pantalla de inicio.
Postcondiciones	Disposición de credenciales.	
Excepciones	Paso	Acción
	1.0	El usuario introduce mal su confirmación de contraseña.
	1.1	El sistema le indica que está mal el campo para que este repita la acción.

Tabla 5.1: CU-001: Registrarse.

CU-002: Iniciar sesión.		
Descripción	Rellena los campos de usuario y contraseña.	
Actores	Usuario.	
Precondiciones	Dispone de las credenciales necesarias.	
Flujo normal	Paso	Acción
	1.	El usuario pulsa en *Iniciar Sesión* en la página principal.
	2.	El usuario rellena los campos usuario y contraseña y es devuelto a la pantalla de inicio.
Postcondiciones	Está reconocido como administrador	
Excepciones	Paso	Acción
	1.0	El usuario introduce mal su contraseña.
	1.1	El sistema indica que la contraseña es incorrecta.
	2.0	El usuario introduce un usuario erróneo.
	2.1	El sistema muestra que el usuario no existe.

Tabla 5.2: CU-002: Iniciar sesión.

CU-003: Cerrar sesión.		
Descripción	El usuario cierra la sesión actual.	
Actores	Administrador.	
Precondiciones	El usuario ha iniciado sesión.	
Flujo normal	Paso	Acción
	1.	El usuario pulsa su nombre para abrir el menú de Administrador.
	2.	El usuario pulsa sobre *Cerrar sesión* y es devuelto a la pantalla de inicio.
Postcondiciones	La sesión esta cerrada.	
Excepciones	Paso	Acción
	-	-

Tabla 5.3: CU-003: Cerrar sesión.

CU-004: Cambiar contraseña.		
Descripción	El usuario introduce su nueva contraseña en el sistema.	
Actores	Administrador.	
Precondiciones	El usuario ha iniciado sesión.	
Flujo normal	Paso	Acción
	1.	El usuario pulsa su nombre para abrir el menú de Administrador.
	2.	El usuario introduce su antigua contraseña, su nueva contraseña y la confirmación de su nueva contraseña.
	3.	El usuario pulsa *Cambiar contraseña.*
Postcondiciones	El usuario dispone de una nueva contraseña.	
Excepciones	Paso	Acción
	1.0	La contraseña es incorrecta.
	1.1	El sistema le comunica al usuario que ha escrito una contraseña incorrecta.
	2.0	La confirmación de contraseña es incorrecta.
	2.1	El sistema indica al usuario que ambas contraseñas no coinciden.

Tabla 5.4: CU-004: Cambiar contraseña.

5.2 Administrador

En esta sección se describen los casos de usos correspondientes unicamente a los administradores de la web.

5.2.1 Añadir recursos

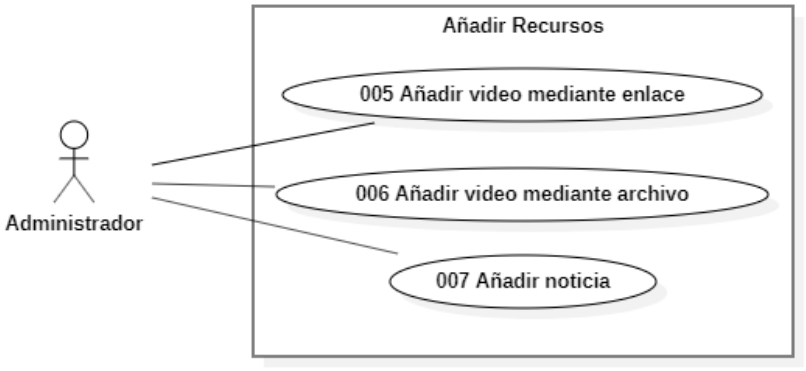

Figura 5.2: Diagrama Administrador-Añadir Recursos

CU-005: Añadir vídeo mediante enlace.		
Descripción	El Administrador sube un vídeo al sistema indicando el enlace, una miniatura y su nombre.	
Actores	Administrador.	
Precondiciones	El Administrador ha iniciado sesión.	
Flujo normal	Paso	Acción
	1.	El Administrador pulsa su nombre para abrir el menú de Administrador y selecciona *Añadir Link(vídeo)*.
	2.	El Administrador introduce un enlace válido, un nombre para el vídeo y una miniatura(subida desde su sistema de ficheros). También indica si este estará a la vista u oculto.
	3.	El Administrador pulsa en *Añadir Vídeo*.
Postcondiciones	El vídeo es subido a la plataforma.	
Excepciones	Paso	Acción
	1.0	El enlace no es válido.
	1.1	El sistema le comunica al Administrador que el enlace no puede ser reproducido.
	2.0	El vídeo ya existe.
	2.1	El sistema le comunica al Administrador que ese vídeo ya ha sido subido anteriormente.
	3.0	Algún campo queda sin completar.
	3.1	El sistema indica al usuario que debe completar todos los campos.

Tabla 5.5: CU-005: Añadir vídeo mediante enlace.

CU-006: Añadir vídeo mediante archivo.		
Descripción	El administrador sube un vídeo al sistema desde su archivos, también sube una miniatura e indica su nombre.	
Actores	Administrador.	
Precondiciones	El Administrador ha iniciado sesión.	
Flujo normal	Paso	Acción
	1.	El Administrador pulsa su nombre para abrir el menú de Administrador y selecciona *Añadir Vídeo*.
	2.	El Administrador sube desde su sistema de archivos un vídeo en el formato preestablecido y una imagen correspondiente a la miniatura que luego identificará dicho vídeo. A su vez indica un nombre por el cual se reconocerá el vídeo. También indica si este estará a la vista u oculto.
	3.	El Administrador pulsa en *Añadir Vídeo*.
Postcondiciones	El vídeo es subido a la plataforma.	
Excepciones	Paso	Acción
	1.0	Algún archivo no es válido.
	1.1	El sistema le comunica al Administrador que archivo no es válido para que lo rectifique.
	2.0	El vídeo ya existe.
	2.1	El sistema le comunica al Administrador que ese archivo ya ha sido subido anteriormente.
	3.0	Algún campo queda sin completar.
	3.1	El sistema indica al usuario que debe completar todos los campos.

Tabla 5.6: CU-006: Añadir vídeo mediante archivo.

CU-007: Añadir noticia.		
Descripción	El administrador rellena los campos necesarios para añadir una noticia.	
Actores	Administrador.	
Precondiciones	El Administrador ha iniciado sesión.	
Flujo normal	Paso	Acción
	1.	El Administrador pulsa su nombre para abrir el menú de Administrador y selecciona *Añadir Noticia*.
	2.	El Administrador introduce un titular, un cuerpo en formato *html* y hasta 3 etiquetas que identifican la noticia. También indica si esta estará a la vista u oculta.
	3.	El Administrador pulsa en *Añadir Noticia*.
Postcondiciones	La noticia es subida a la plataforma.	
Excepciones	Paso	Acción
	1.0	Algún campo queda sin completar.
	1.1	El sistema indica al usuario que debe completar todos los campos.
	2.0	La noticia ya existe.
	2.1	El sistema le comunica al Administrador que dicha noticia ya existe en el sistema.

Tabla 5.7: CU-007: Añadir noticia.

5.2.2 Edición

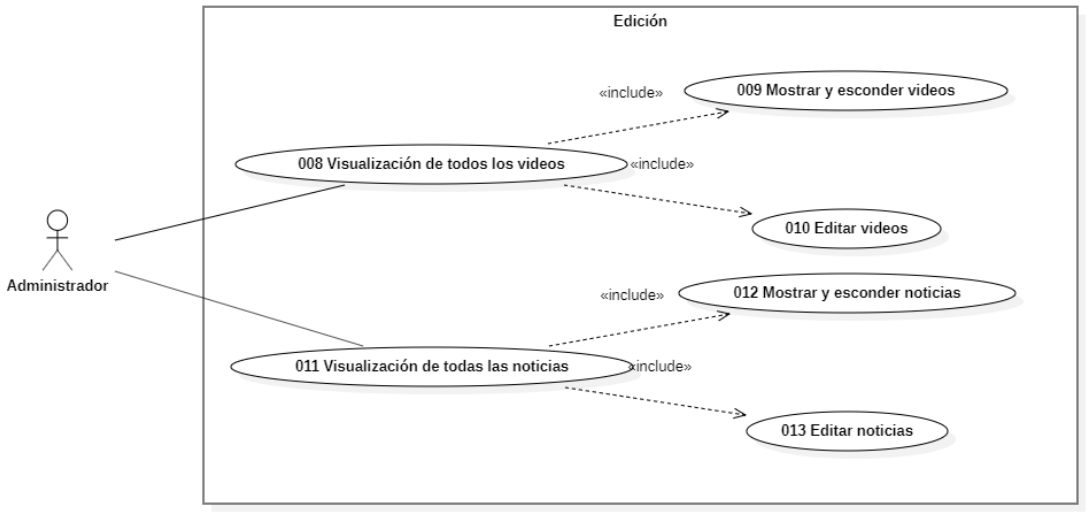

Figura 5.3: Diagrama Administrador-Edición

CU-008: Visualización de todos los vídeos.		
Descripción	El Administrador puede ver la lista completa de vídeos que alberga la plataforma.	
Actores	Administrador.	
Precondiciones	El Administrador ha iniciado sesión.	
Flujo normal	Paso	Acción
	1.	El Administrador pulsa su nombre para abrir el menú de Administrador y selecciona *Editar Vídeos*.
Postcondiciones	Tiene a la vista todos los vídeos.	
Excepciones	Paso	Acción
	-	-

Tabla 5.8: CU-008: Visualización de todos los vídeos.

CU-009: Mostrar y esconder vídeos.		
Descripción	El Administrador puede elegir la visibilidad de los vídeos.	
Actores	Administrador.	
Precondiciones	El Administrador ha accedido a la pantalla de edición de vídeos.	
Flujo normal	Paso	Acción
	1.	El Administrador pulsa sobre la *Checkbox Esconder* del vídeo que quiere mostrar u ocultar al usuario estándar.
Postcondiciones	El vídeo cambia su estado al deseado.	
Excepciones	Paso	Acción
	-	-

Tabla 5.9: CU-009: Mostrar y esconder vídeos

CU-010: Editar vídeos		
Descripción	El Administrador puede editar el nombre, la miniatura y el estado(oculto) del vídeo.	
Actores	Administrador.	
Precondiciones	El Administrador ha accedido a la pantalla de edición de vídeos.	
Flujo normal	Paso	Acción
	1.	El Administrador pulsa sobre el vídeo que desea editar.
	2.	El Administrador puede editar el nombre, la imagen o el estado del vídeo.
	3.	El Administrador pulsa sobre *Editar Vídeo*.
Postcondiciones	El vídeo queda editado.	
Excepciones	Paso	Acción
	1.0	El formato no es válido.
	1.1	El sistema informa de que el formato de la nueva miniatura no es válido.

Tabla 5.10: CU-010: Editar vídeos

CU-011: Visualización de todas las noticias.		
Descripción	El Administrador puede ver la lista completa de noticias que alberga la plataforma.	
Actores	Administrador.	
Precondiciones	El Administrador ha iniciado sesión.	
Flujo normal	Paso	Acción
	1.	El Administrador pulsa su nombre para abrir el menú de Administrador y selecciona *Editar Noticias*.
Postcondiciones	Tiene a la vista todas las noticias.	
Excepciones	Paso	Acción
	-	-

Tabla 5.11: CU-011: Visualización de todas las noticias.

CU-012: Mostrar y esconder noticias.		
Descripción	El Administrador puede elegir la visibilidad de las noticias.	
Actores	Administrador.	
Precondiciones	El Administrador ha accedido a la pantalla de edición de noticias.	
Flujo normal	Paso	Acción
	1.	El Administrador pulsa sobre la *Checkbox Esconder* de la noticia que quiere mostrar u ocultar al usuario estándar.
Postcondiciones	La noticia cambia su estado al deseado.	
Excepciones	Paso	Acción
	-	-

Tabla 5.12: CU-012: Mostrar y esconder noticias.

CU-013:Editar noticias.		
Descripción	El Administrador puede editar el nombre, el cuerpo, el estado(oculto) y las etiquetas de la noticia.	
Actores	Administrador.	
Precondiciones	El Administrador ha accedido a la pantalla de edición de noticias.	
Flujo normal	Paso	Acción
	1.	El Administrador pulsa sobre la noticia que desea editar.
	2.	El Administrador puede editar el nombre, el cuerpo, estado y las etiquetas de la noticia. También dispone de una preview del cuerpo de esta.
	3.	El Administrador pulsa sobre *Editar noticia.*
Postcondiciones	La noticia es editada.	
Excepciones	Paso	Acción
	1.0	Campos incompletos
	1.1	El sistema informa de los campos que no han sido rellenados.

Tabla 5.13: CU-013: Editar noticias.

5.3 Usuario Estándar

Aquí se representan los casos de uso de los que dispondrá el usuario que visite la plataforma para informarse sobre la empresa.

5.3.1 Página Principal

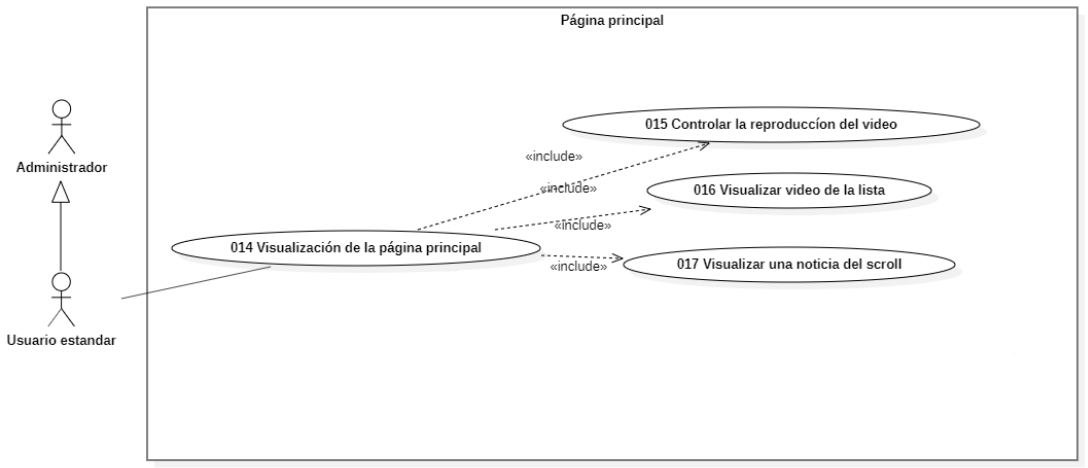

Figura 5.4: Diagrama Usuario Estándar-Página Principal

CU-014: Visualización de la página principal.		
Descripción	Se dispondrá de un vídeo principal, una lista de los vídeo en la plataforma, un *scroll* de noticias y una selección de etiquetas a la vista del usuario.	
Actores	Usuario estándar.	
Precondiciones	-	
Flujo normal	Paso	Acción
	1.	El Usuario entra en la plataforma o pulsa la tecla *Inicio*.
Postcondiciones	Tiene a la vista la página principal.	
Excepciones	Paso	Acción
	-	-

Tabla 5.14: CU-014: Visualización de la página principal.

CU-015: Controlar la reproducción del vídeo.		
Descripción	El usuario puede hacer uso de los controles habituales de un reproductor de vídeo.	
Actores	Usuario estándar.	
Precondiciones	Hay un vídeo cargado	
Flujo normal	Paso	Acción
	1.	El usuario interactúa con los botones.
Postcondiciones	-	
Excepciones	Paso	Acción
	-	-

Tabla 5.15: CU-015: Controlar la reproducción del vídeo.

CU-016: Visualizar vídeo de la lista.		
Descripción	El usuario tiene la posibilidad de ver el vídeo que desea entre los que aparecen en la lista.	
Actores	Usuario estándar.	
Precondiciones	La lista tiene vídeos.	
Flujo normal	Paso	Acción
	1.	El Usuario pulsa sobre el vídeo que desea ver y este se carga.
Postcondiciones	El vídeo se reproduce.	
Excepciones	Paso	Acción
	-	-

Tabla 5.16: CU-016: Visualizar vídeo de la lista.

CU-017: Visualizar una noticia del *scroll*.		
Descripción	El usuario tiene la posibilidad de ver la noticia que desea entre las que aparecen en el scroll.	
Actores	Usuario estándar.	
Precondiciones	El *scroll* tiene noticias.	
Flujo normal	Paso	Acción
	1.	El Usuario pulsa sobre la noticia que sedea ver y esta se abre sustituyendo al pantalla de inicio.
Postcondiciones	Se visualiza la noticia.	
Excepciones	Paso	Acción
	-	-

Tabla 5.17: CU-017: Visualizar una noticia del *scroll*.

5.3.2 Página Noticias

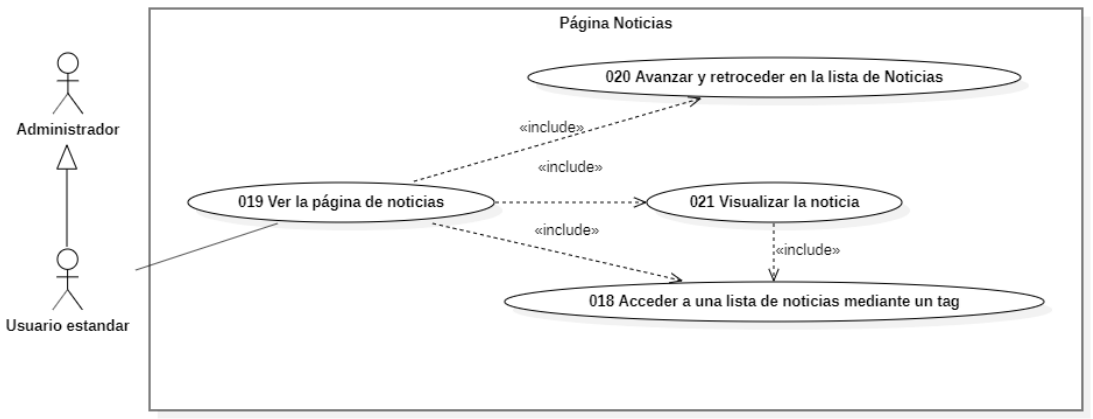

Figura 5.5: Diagrama Usuario Estándar-Página Noticias

CU-018: Acceder a lista de noticias mediante ua etiqueta.		
Descripción	El usuario tiene la posibilidad de acceder a una lista de noticias mediante una etiqueta.	
Actores	Usuario estándar.	
Precondiciones	Existen noticias con alguna etiqueta.	
Flujo normal	Paso	Acción
	1.	El Usuario pulsa sobre la *etiqueta* deseado y se abre una lista de noticias con esa etiqueta.
Postcondiciones	Se visualiza una lista de noticias con la etiqueta deseado.	
Excepciones	Paso	Acción
	1.0	No hay noticias con la etiqueta.
	1.1	El sistema avisa al usuario de que dicha etiqueta no tiene noticias asociadas.

Tabla 5.18: CU-018: Acceder a lista de noticias mediante una etiqueta.

CU-019: Ver la página de noticias.		
Descripción	Se visualiza una lista de noticias con posibilidad de avanzar y retroceder en ella. Además se visualiza la lista de las etiquetas de noticias.	
Actores	Usuario estándar.	
Precondiciones	Hay noticias en el sistema	
Flujo normal	Paso	Acción
	1.	El Usuario entra en la vista pulsando la tecla *Noticias*.
Postcondiciones	Tiene a la vista la página de noticias.	
Excepciones	Paso	Acción
	-	-

Tabla 5.19: CU-019: Ver la página de noticias.

CU-020: Avanzar u retroceder en la lista de noticias.		
Descripción	El usuario puede avanzar o retroceder en la lista.	
Actores	Usuario estándar.	
Precondiciones	Existen noticias suficientes para avanzar o retroceder	
Flujo normal	Paso	Acción
	1.	El Usuario pulsa en la tecla *Siguiente* o *Anterior*.
Postcondiciones	Tiene a la vista la página de noticias.	
Excepciones	Paso	Acción
	1.0	No hay suficientes noticias.
	1.1	El botón aparece en gris y no se puede pulsar.

Tabla 5.20: CU-020: Avanzar u retroceder en la lista de noticias.

CU-021: Visualizar la noticia.		
Descripción	El usuario puede ver la noticia con sus etiquetas correspondientes.	
Actores	Usuario estándar.	
Precondiciones	La noticia existe.	
Flujo normal	Paso	Acción
	1.	El Usuario pulsa sobre la noticia.
Postcondiciones	Se visualiza la noticia	
Excepciones	Paso	Acción
	-	-

Tabla 5.21: CU-021: Visualizar la noticia.

5.3.3 Extras

CU-022: Visualización de una nube de etiquetas.		
Descripción	El usuario visualiza una nube con las etiquetas mas usadas.	
Actores	Usuario estándar.	
Precondiciones	Existen noticias con alguna etiqueta.	
Flujo normal	Paso	Acción
	1.	El usuario entra en la página de *Noticias*
Postcondiciones	Se visualiza una lista de etiquetas al que usuario puede acceder.	
Excepciones	Paso	Acción
	1.0	No hay noticias con etiquetas.
	1.1	No hay nube de etiquetas.

Tabla 5.22: CU-022: Visualización de una nube de etiquetas.

CU-023: Acceder a la lista total de etiquetas.		
Descripción	El usuario visualiza una página de etiquetas, puede navegar por ellas y acceder a la deseada.	
Actores	Usuario estándar.	
Precondiciones	Existen noticias con alguna etiqueta.	
Flujo normal	Paso	Acción
	1.	El usuario selecciona el botón *+más etiquetas* de la *nube de etiquetas*
Postcondiciones	Se visualiza una lista de etiquetas al que usuario puede acceder.	
Excepciones	Paso	Acción
	1.0	No hay noticias con etiquetas.
	1.1	No hay lista de etiquetas.

Tabla 5.23: CU-023: Acceder a la lista total de etiquetas.

Capítulo 6

Diseño de la aplicación

El diseño de software es el proceso por el que un agente crea una especificación de un artefacto de software, pensado para cumplir unos objetivos, utilizando un conjunto de componentes primitivos y sujeto a restricciones.[4] Para el diseño y proceso de esta aplicación se han utilizado como base las diapositivas de la asignatura *Programación Avanzada* [5], las cuales, me han ayudado a comprender el uso las tecnologías utilizadas para poder llevar a cabo su uso.

6.1 Patrones

Un diseñador de software o arquitecto puede identificar un problema de diseño que ha sido visitado y quizás incluso solucionado por otros anteriormente. La plantilla o patrón que describe una solución a un problema común se conoce como patrón de diseño. La reutilización de tales patrones puede ayudar a agilizar el proceso de desarrollo del software.[6]

6.1.1 Modelo-Vista-Controlador

El patrón **MVC (Modelo-Vista-Controlador)**[7] se encarga de separar los datos, la lógica de la aplicación y la interfaz de usuario en tres componentes aislados.

El **Modelo** contiene la representación de las entidades, su lógica de negocio y su almacén de datos. Sus funciones son:

- Acceso a la capa de almacenamiento de datos.
- Definición de las reglas de negocio (funcionalidades del sistema).
- Registro de las vistas y controladores del sistema.

El **Controlador** actúa como intermediario entre el *modelo* y la *vista*, de manera que gestiona el flujo de información entre ellos y las transformaciones para adaptar los datos entre ambos.

Sus funciones son:

- Recibir los eventos de entrada.
- Contener las reglas de gestión de eventos. Como es por ejemplo, si se puede una noticia con un *id*, devolver dicha noticia hacia la interfaz de usuario.

La **Vista** es el interfaz de usuario. Se compone de la información que se envía al cliente y los mecanismos de interacción con éste.

Sus funciones son:

- Recibir datos del modelo y mostrárselos al usuario.
- Tienen un registro de su controlador asociado.
- Pueden dar el servicio de *"Actualización()"*, para que sea invocado por el controlador o por el modelo.

El flujo habitual que sigue este patrón lo podemos observar en la figura 6.1

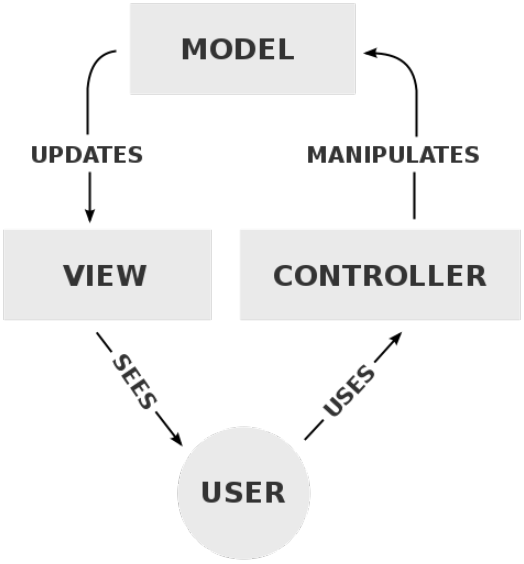

Figura 6.1: Flujo MVC By:Regis Frey

6.1.2 Data Access Object (DAO)

DAO (objeto de acceso a datos) es un componente de software que suministra una interfaz común entre la aplicación y uno o más dispositivos de almacenamiento de datos, tales como una base de datos o un archivo. Con ello conseguimos el acceso a los datos pero sin exponer los detalles de la base de datos. Este aislamiento es compatible con el principio de la responsabilidad individual, que establece que cada módulo o clase debe tener una sola parte de la funcionalidad proporcionada por el software, y que dicha responsabilidad debe ser totalmente encapsulada por la clase.

Pese a que este patrón de diseño es igualmente aplicable para la mayoría de lenguajes, áste se asocia tradicionalmente a **Java EE** aplicaciones y bases de datos relacionales, que es lo que se utiliza en el proyecto.

El patrón **DAO** propone separar por completo la lógica de negocio de la lógica de acceder a los datos. En nuestro caso, se aplica creando el **DAO** de cada entidad dando así las posibilidades de llamar a esa interfaz a la hora de manejar las entidades en los servicios.

El diseño básico de **DAO** es el que se muestra en la figura 6.2.

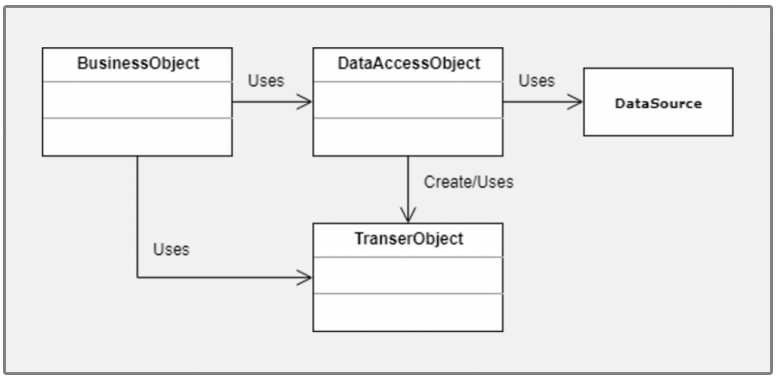

Figura 6.2: DAO

6.1.3 Fachada

El patrón **Fachada** viene motivado por la necesidad de estructurar un entorno de programación y reducir su complejidad con la división en subsistemas, minimizando las comunicaciones y dependencias entre éstos.

Este patrón se puede aplicar cuando se necesite proporcionar una interfaz simple para un subsistema complejo, o cuando se quiera estructurar varios subsistemas en capas, haciendo fachadas como punto de entrada de cada nivel. Éste es nuestro escenario ya que disponemos de distintos niveles como pueden ser los *DAO*, que tienen sus propias interfaces, los servicios, o mismo el propio controlador.

Además también nos permite ocultar la complejidad de interactuar con un conjunto de subsistemas, proporcionando una interfaz de alto nivel, la cual se encarga de realizar la comunicación con todos los subsistemas necesarios (figura 6.3). La fachada es una buena estrategia cuando requerimos interactuar con varios subsistemas para realizar una operación concreta ya que se necesita tener el conocimiento técnico y funcional para saber qué operaciones de cada subsistema tenemos que ejecutar y en qué orden, lo que puede resultar muy complicado cuando los sistemas empiezan a crecer demasiado[8].

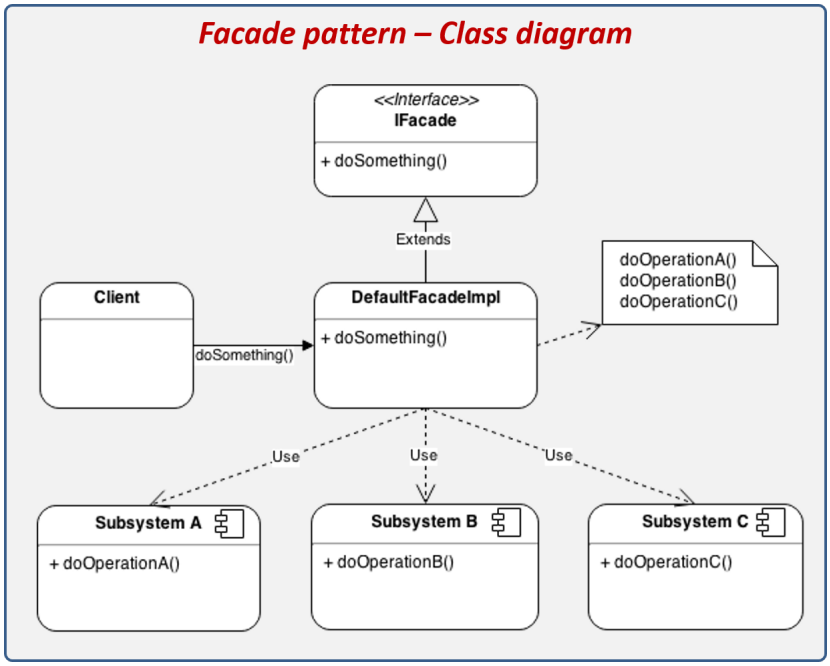

Figura 6.3: Diagrama fachada

6.2 Arquitectura

La arquitectura del proyecto tiene una división clara entre lo que sería el **BackEnd**, compuesto del modelo y de los controladores, y el **FrontEnd** que solo contendría la interfaz. Esta división es debida a que como interfaz se ha elegido el uso de **React**, que contiene una organización propia de ficheros.

6.2.1 Arquitectura Backend

Se divide en los siguientes ficheros principales:

- backend/src

- backend/target : directorio donde se guardan las clases compiladas de la aplicación.
- backend/pom.xml: archivo utilizado por **Maven** para describir el proyecto, sus dependencias, los componentes externos y el orden de ejecución.
- backend/archivos: Aquí es donde se guardan los vídeos subidos a la plataforma, y también dispone de una carpeta para las imágenes, en caso de querer cambiar el sistema de guardado.
- backend/.settings
- backend/node_modules

BackEnd/src

La carpeta **src** podríamos decir que es la carpeta principal del proyecto y en ella se almacena lo siguiente:

- /src/main
- /src/sql: En este directorio se almacenan los archivos donde se declaran las bases de datos y las posibles inserciones iniciales.
- /src/test. Esta carpeta se divide en */java* donde están los propios test unitarios creados con **JUnit** y */resources* que es donde se almacenan los recursos de dichos test.

La carpeta **main** consta de dos directorios. Por un lado tenemos **/resources** donde se almacenan los recursos de la aplicación como puede ser los mensajes que lanzan las excepciones cuando salen por los controladores.

Por el otro tenemos **/java** que es donde se encuentra el código del **BackEnd** de la aplicación. Este se divide a su vez en dos directorios.

- Model: Aquí estan contenidos los paquetes que contienen las **entidades** y sus correspondientes **DAO**, las **excepciones** y las interfaces de los **servicios**.
- REST: En el directorio rest tenemos los **DTO** que se encargan de sustituir a las entidades con sus respectivos conversores, los propios **controladores** y una carpeta **common** donde se configuran las diferentes posibilidades que nos brinda el Framework **Spring** (como pueden ser *security* o *JWT*).

6.2.2 Arquitectura FrontEnd

Se divide en tres ficheros:

- frontend/node_modules: Contiene los módulos de la interfaz como pueden ser *boostrap, react, babel, css...*
- frontend/public: Contiene imágenes o iconos personalizados que se ven reflejados por ejemplo, en la pestaña o en la barra de navegación. Se han dejado por defecto las de react.
- frontend/package.json: contiene metadatos sobre su aplicación o módulo, así como la lista de dependencias para instalar desde *npm* cuando se ejecuta *npm install*.
- frontend/src

FrontEnd/src

Se divide en los siguientes ficheros:

- backend: Aquí es donde se da forma a la capa de acceso a servicios. Es donde están las funciones que sirven para hacer peticiones a nuestros controladores.
- i18n: Directorio donde se almacenan los ficheros de mensajes y su internacionalización.
- modules: Directorio con la capa de interfaz de usuario.
- polyfills: Directorio donde se hacen las importaciones de los *Polyfills*, que son fragmentos de código que proporciona la tecnología que el desarrollador espera que el navegador proporcione de forma nativa.
- store: Creación de la *store*, que es como un almacenamiento de datos durante la ejecución utilizado por *react* para el uso de los estados.
- index.js
- styles.css

6.3 Capa modelo

En la capa modelo tenemos las entidades con sus respectivos *DAO* y los servicios que se encargan de implementar los casos de uso.

6.3.1 Base de datos

El primer paso para poder entender como funcionan las entidades es observar el modelo relacional de la figura 6.4. En el podemos apreciar lo siguiente:

- **user** que es la que se encarga de guardar los usuarios. Esta entidad no tiene ninguna relación con el resto.

- **report** tiene tres relaciones, que representan cada uno de las etiquetas que puede tener una noticia. Estas etiquetas son opcionales y una etiqueta puede estar en noticias distintas.

- **vídeo** es la más compleja. Dispone de dos relaciones, pero en la lógica de negocio, esto adquiere aún más restricciones. Un vídeo puede tener un **videofile** o no, sin embargo, si tiene un **videofile** no puede tener un link y viceversa. Dado que es la manera de diferenciar los dos tipos de vídeos que almacenamos. Por otro lado, un vídeo siempre debe tener una imagen asociada, su miniatura. Pero esta imagen puede ser utilizada por diversos vídeos.

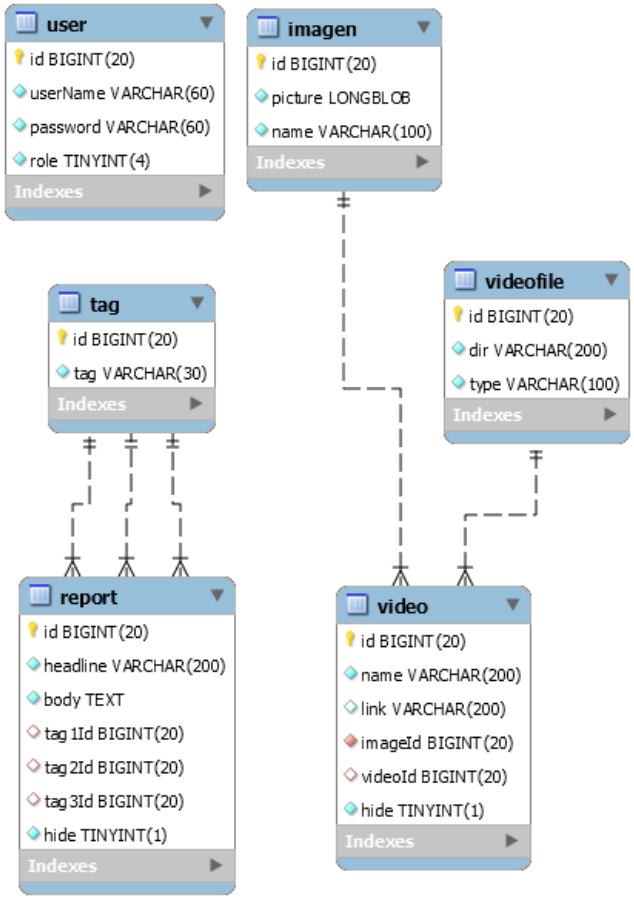

Figura 6.4: Modelo E-R

Para crear las tablas correspondientes al modelo relacional visto en la figura 6.4 se ha utilizado el código del listado 6.1.

```
CREATE TABLE IF NOT EXISTS User (
    id BIGINT NOT NULL AUTO_INCREMENT,
    userName VARCHAR(60) COLLATE latin1_bin NOT NULL,
    password VARCHAR(60) NOT NULL,
    role TINYINT NOT NULL,
    CONSTRAINT UserPK PRIMARY KEY (id),
    CONSTRAINT UsernameUniqueKey UNIQUE (userName)
) ENGINE = InnoDB;

CREATE TABLE IF NOT EXISTS Imagen (
    id BIGINT NOT NULL AUTO_INCREMENT,
    picture LONGBLOB NOT NULL,
    name VARCHAR(100) NOT NULL,
    CONSTRAINT ImagenPK PRIMARY KEY (id),
    CONSTRAINT NameImagenUniqueKey UNIQUE (name)
) ENGINE = InnoDB;


CREATE TABLE IF NOT EXISTS VideoFile (
    id BIGINT NOT NULL AUTO_INCREMENT,
    dir VARCHAR(200) NOT NULL,
    type VARCHAR(100) NOT NULL,
    CONSTRAINT VideoPK PRIMARY KEY (id),
    CONSTRAINT DirUniqueKey UNIQUE (dir)
) ENGINE = InnoDB;

CREATE TABLE IF NOT EXISTS Video (
    id BIGINT NOT NULL AUTO_INCREMENT,
    name VARCHAR(200) NOT NULL,
    link VARCHAR(200),
    imageId BIGINT NOT NULL,
    videoId BIGINT,
    hide Boolean NOT NULL,
    CONSTRAINT UserPK PRIMARY KEY (id),
    CONSTRAINT LinkUniqueKey UNIQUE (link),
    CONSTRAINT VideoImageId FOREIGN KEY(imageId)
        REFERENCES Imagen (id),
    CONSTRAINT VideoVideoId FOREIGN KEY(videoId)
        REFERENCES VideoFile (id)
) ENGINE = InnoDB;

CREATE TABLE IF NOT EXISTS Tag (
    id BIGINT NOT NULL AUTO_INCREMENT,
```

```
    tag VARCHAR(30) NOT NULL,
    CONSTRAINT TagPK PRIMARY KEY (id),
    CONSTRAINT TagUniqueKey UNIQUE (tag)
) ENGINE = InnoDB;

CREATE TABLE IF NOT EXISTS Report (
    id BIGINT NOT NULL AUTO_INCREMENT,
    headline VARCHAR(200) NOT NULL,
    body text NOT NULL,
    tag1Id BIGINT,
    tag2Id BIGINT,
    tag3Id BIGINT,
    hide Boolean NOT NULL,
    CONSTRAINT ReportPK PRIMARY KEY (id),
    CONSTRAINT HeadlineUniqueKey UNIQUE (headline),
    CONSTRAINT ReportTag1IdFK FOREIGN KEY(tag1Id)
        REFERENCES Tag (id),
    CONSTRAINT ReportTag2IdFK FOREIGN KEY(tag2Id)
        REFERENCES Tag (id),
    CONSTRAINT ReportTag3IdFK FOREIGN KEY(tag3Id)
        REFERENCES Tag (id)
) ENGINE = InnoDB;
```

Listado 6.1: Archivo de craci´on de la BD

Gracias a utilizar el predicado *IF NOT EXISTS* podemos borrar tablas de manera que no se tenga que generar todo de nuevo y perder las que no queremos borrar. Por ejemplo, si queremos vaciar los vídeos, borramos las tablas correspondientes con sus respectivas restricciones y ejecutamos de nuevo el archivo de creación.

6.3.2 Entidades

En la figura 6.5 podemos observar las entidades **Report**, **Tag** Y **TagCloud**. Además podemos observar que tanto **Report** como **Tag** tienen DAOs personalizados para llevar a cabo funciones fuera del **JPA**.

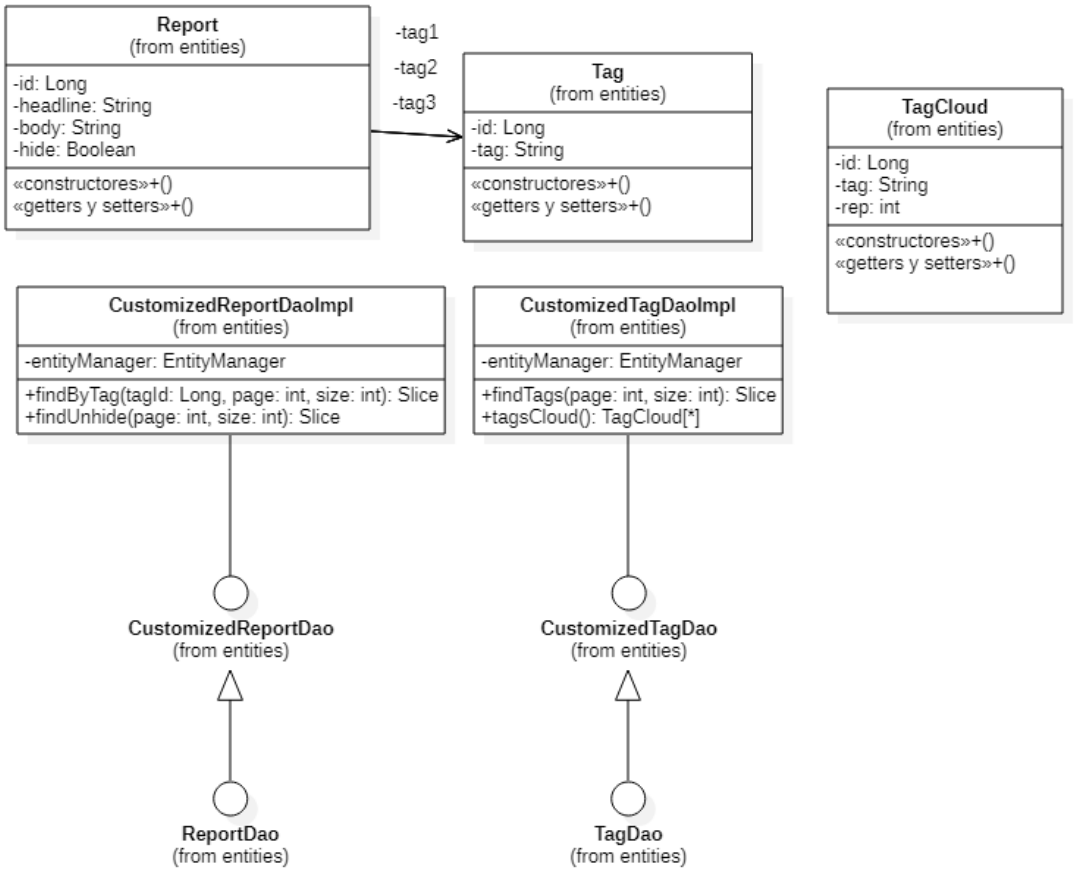

Figura 6.5: Entidades 1

- Report: Consta de un id, una cabecera, un cuerpo y un campo *"hide"* que representa si está o no oculto. Esta entidad representa las noticias.
- Tag: Consta de un id y un nombre que hace referencia a la etiqueta que representa. Una noticia puede contener hasta 3 *tags*.
- TagDao: Se utiliza para representar un **Tag** con un contador de apariciones. Éste nos sirve para crear una nube de etiquetas en el interfaz de usuario. Tiene su propia función personalizada dentro del *DAO* de **Tag**.

En la figura 6.6 observamos la entidad **Vídeo** que contiene una **Imagen** y puede contener un **VideoFile**.

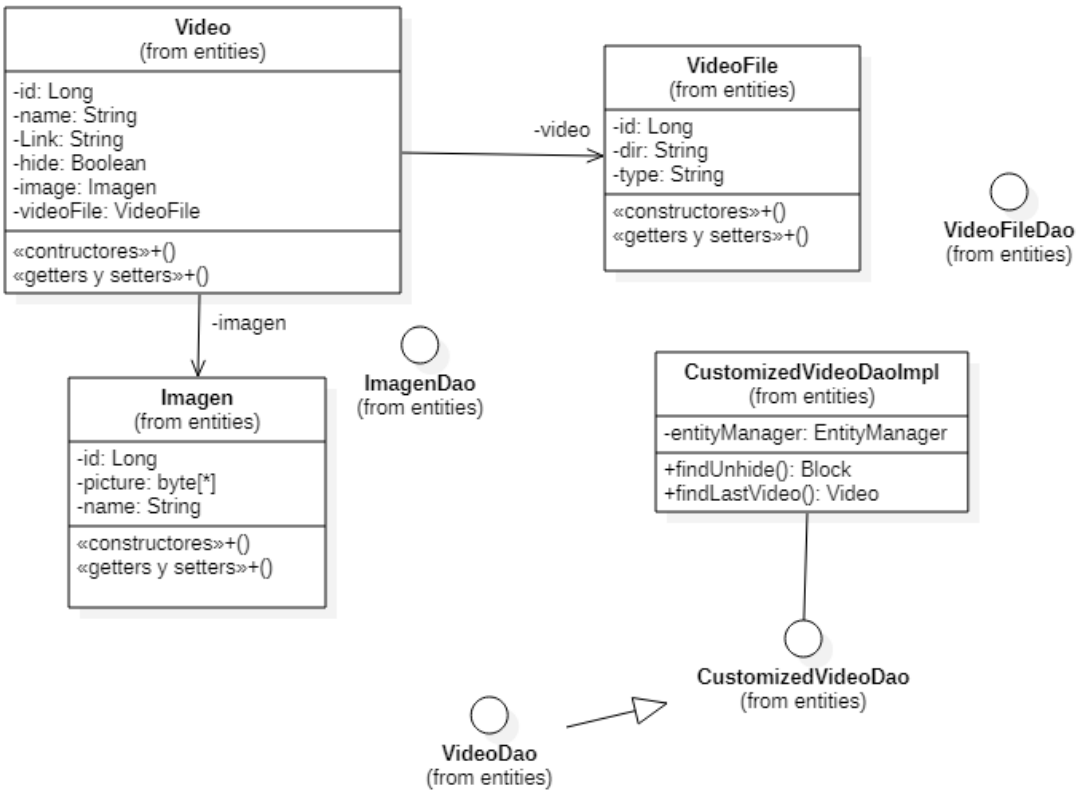

Figura 6.6: Entidades 2

- Video: Tiene un *id*, un nombre, un link en caso de ser un vídeo de otra plataforma, y un *Boolean* que marca su visibilidad. Además este tiene una **Imagen** asociada que sirve como miniatura y puede tener un **VideoFile** en caso de que este sea un vídeo directamente subido a la plataforma.
- Imagen: Consta de un *id*, un *array* de *bytes* y un nombre. Representa, como ya se dijo antes, la miniatura del vídeo.
- VideoFile: Consta de un *id*, un directorio donde se guarda y un tipo, que corresponde con el tipo de archivo. Representa los vídeos que son subidos mediante archivo a la plataforma.

Como se puede observar, los vídeos se guardan en un directorio y se almacena en la base de datos la ruta. Sin embargo, para explorar más posibilidades, las imágenes se almacenan directamente en la base de datos. Esta opción es viable porque las imágenes son miniaturas y de reducido tamaño y calidad.

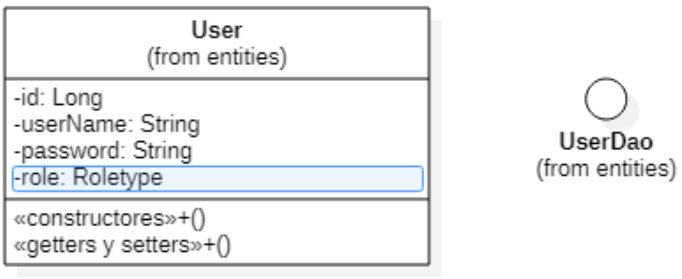

Figura 6.7: Entidades 3

Por último, en la figura 6.7 podemos observar la entidad **User**, que contiene un *id*, un nombre de usuario, una contraseña y un rol. Es este caso solo utilizamos un rol porque solo tenemos un tipo de usuario, pero en caso de tener más tipos, tan solo tendríamos que añadirlo.

6.3.3 Servicios

Los servicios son los que se encargan de llevar a cabo los casos de uso relacionados con las entidades. En esta caso a mayores, se ha creado un objeto llamado **Block** (figura 6.8) que sirve de contenedor de objetos y a su vez de ayuda para realizar peticiones paginadas. Esto es, se permite pedir un número determinado de objetos a partir de una determinada posición y saber si existen más. En este caso usaremos esto para pedir los objetos de n en n posiciones. Por ejemplo, pedimos 10 noticias y habilitamos para pedir las 10 siguientes. También podemos apreciar el objeto **VideoFileAux** que sirve para convertir el formato array, de los vídeos que se han subido a la plataforma al formato *Resource* para enviarlo a la interfaz.

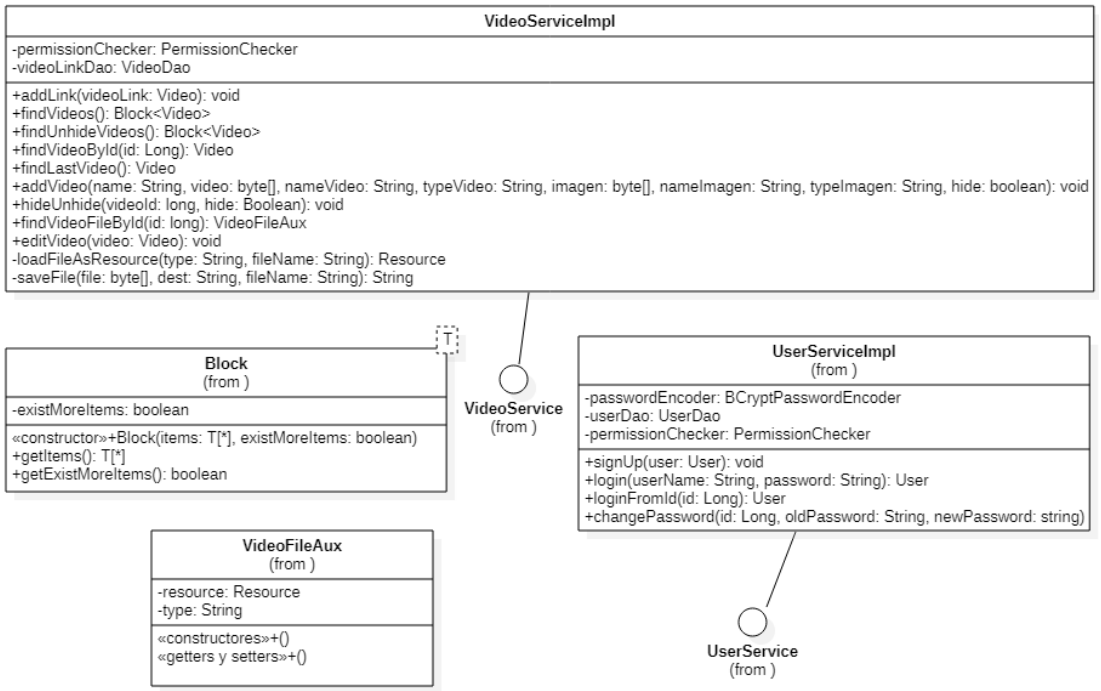

Figura 6.8: Servicios

En la figura 6.8 podemos ver las interfaces **VideoService** y **UserService** con sus respectivas implementaciones.

VideoService

Como vemos en la figura 6.8 la interfaz **VideoService** tiene las siguientes funciones:

- addLink: Se encarga de introducir a la base de datos los vídeos que son el tipo *link* externo.
- findVideos: Devuelve una lista de todos los vídeos. Aunque es este caso está desactivada la función de paginación debido a que no es necesario. En caso de volverse necesaria por la gran cantidad de vídeos en la plataforma se podría activar.
- findUnhideVideos: Devuelve la lista de los vídeos que no estén ocultos. También está desactivada la función de paginación, forzando de manera indirecta a usar la función de ocultar cuando el vídeo sea obsoleto o sobrante.
- findVideoById: Devuelve el vídeo buscado mediante un *id* concreto.
- findLastVideo: Devuelve el último vídeo visible en la plataforma.

- addVideo: Se encarga de añadir un vídeo que se suba a la plataforma en alguno de los formatos permitidos(cualquier formato de video en general).
- findVideoFileById: Devuelve un archivo de vídeo mediante *id*.
- hideUnhide: Cambia el estado de oculto de un vídeo.
- editVideo: Permite modificar el nombre, la miniatura o si está oculto un vídeo.
- loadFileAsResource: Función interna no perteneciente al interfaz que devuelve un vídeo en formato *Resource* a partir de su nombre y su tipo.
- saveFile: Función interna no perteneciente al interfaz que se encarga de guardar el archivo de vídeo en el directorio deseado.

UserService

Como vemos en la figura 6.8 la interfaz **UserService** tiene las siguientes funciones:

- signUp: Permite a un usuario registrarse
- login: Permite a un usuario iniciar sesión.
- loginFromId: Sirve para iniciar sesión tan solo con el *id*. Esto se usará luego para iniciar sesión con un *token*.
- changePassword: Permite cambiar la contraseña si la introducida es correcta.

En la figura 6.9 podemos ver la interfaz **ReportService** con su respectiva implementación y la interfaz auxiliar **PermissionCheker**.

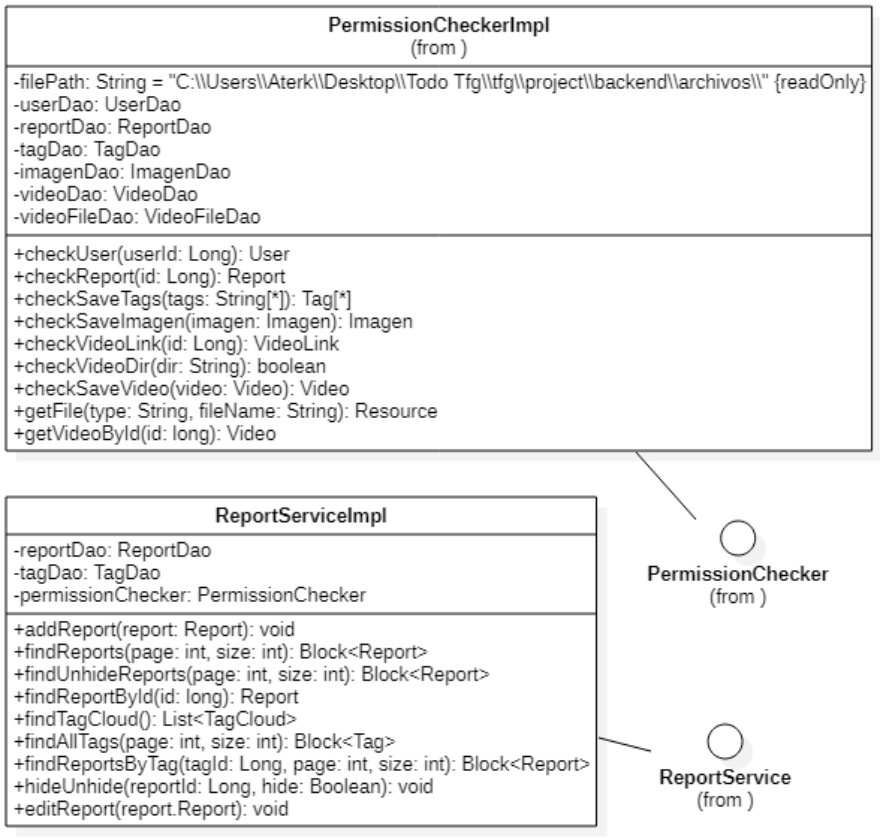

Figura 6.9: Servicios

ReportService

Como vemos en la figura 6.9 la interfaz **ReportService** tiene las siguientes funciones:

- addReport: Añade una noticia a la base de datos con los parámetros indicados.
- findReports: Devuelve una página de la lista de noticias.
- findUnhideReports: Devuelve una página de la lista de noticias que no estén ocultas.
- findReportById: Devuelve una noticia con el *id* introducido.
- findTagCloud: Devuelve una lista de las etiquetas más usadas. En esta caso está configurado para 15 en el *DAO*.
- findAllTags: Devuelve una página de la lista de etiquetas.

- findReportsByTag: Devuelve una lista de las noticias que contienen una etiqueta.
- hideUnhide: Cambia el estado de visibilidad de la noticia.
- editReport: Permite cambiar todas las características de una noticia menos el *id*.

En el caso de las noticias y las etiquetas sí que se ha habilitado la paginación debido a la posibilidad de gran cantidad de noticias almacenables a bajo coste.

PermissionChecker

Como vemos en la figura 6.8 la interfaz **PermissionChecker** tiene las siguientes funciones:

- checkUser: Devuelve un usuario a partir de *id*.
- CheckReport: Devuelve una noticia a partir del *id*.
- CheckSaveTags: Introduciendo 3 etiquetas en formato *String* las inserta en la base de datos en caso de no existir y las devuelve en el formato de la entidad.
- checkSaveImagen: Comprueba si una imagen ya existe en la base de datos, en caso de no existir, la introduce. Devuelve la imagen con su identificador.
- checkVideolink: Devuelve un vídeo a partir de su *id*.
- checkVideoDir: Comprueba si un vídeo ya existe en el directorio.
- checkSaveVideo: Introduciendo un archivo de vídeo, lo guarda y lo devuelve con su identificador.
- getFile: Introduciendo el tipo de archivo y el nombre, lo devuelve en caso de estar guardado en el directorio de archivos. Está preparado para vídeos y fotos en caso de que las pasáramos a guardar en directorio.
- getVideoId: Devuelve un vídeo a partir de su *id*. Este vídeo es del formato *videoFile*.

6.3.4 Excepciones

En el proyecto disponemos de las siguientes excepciones:

- DuplicateInstanceException: Utilizada para cualquier instancia duplicada, ya sea usuario, noticia, vídeo...
- InstanceNotFoundException : Se utiliza cuando un objeto que se pide no existe en la base de datos.
- IncorrectLoginException: Se utiliza para los inicios de sesión incorrectos.
- IncorrectPasswordException: Utilizada en la función de cambio de contraseña cuando la que llega es incorrecta.
- PermissionException: Utilizada para la seguridad del cambio de contraseña.

6.4 Capa REST

6.4.1 Data Transfer Object

Un objeto de transferencia de datos (en inglés, data transfer object, abreviado DTO) es un objeto que transporta datos entre procesos. La motivación de su uso tiene relación con el hecho que la comunicación entre procesos se realiza generalmente mediante interfaces remotas (por ejemplo, servicios web), donde cada llamada es una operación costosa. Como la mayor parte del coste de cada llamada está relacionado con la comunicación de ida y vuelta entre el cliente y servidor, una forma de reducir el número de llamadas es usando un objeto (el DTO) que agrega los datos que habrían sido transferidos por cada llamada, pero que son entregados en una sola llamada. [9, 10]

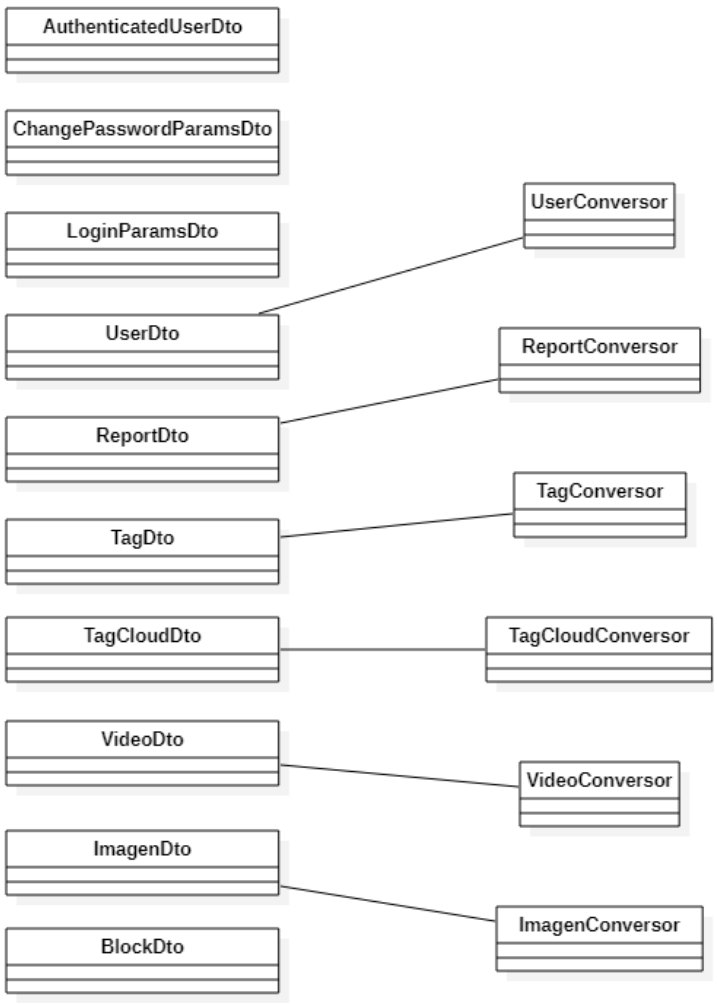

Figura 6.10: Controladores

Como se puede ver en la figura 6.10 en el proyecto tenemos un objeto de transferencia de datos para cada entidad, exceptuando *VideoFile* que no se exporta como tal, sino como un *Resource* ya. Además contamos con unos objetos de transferencia extras como son *AutheticatedUserDto*, *ChangePasswordParamsDto* y *LoginParamsDto*, que se utilizan en la gestión de usuarios.

6.4.2 Controladores

Hay un total de 3 controladores como podemos ver en la figura 6.11. Se encargan de enviar y recibir las peticiones de la interfaz web.

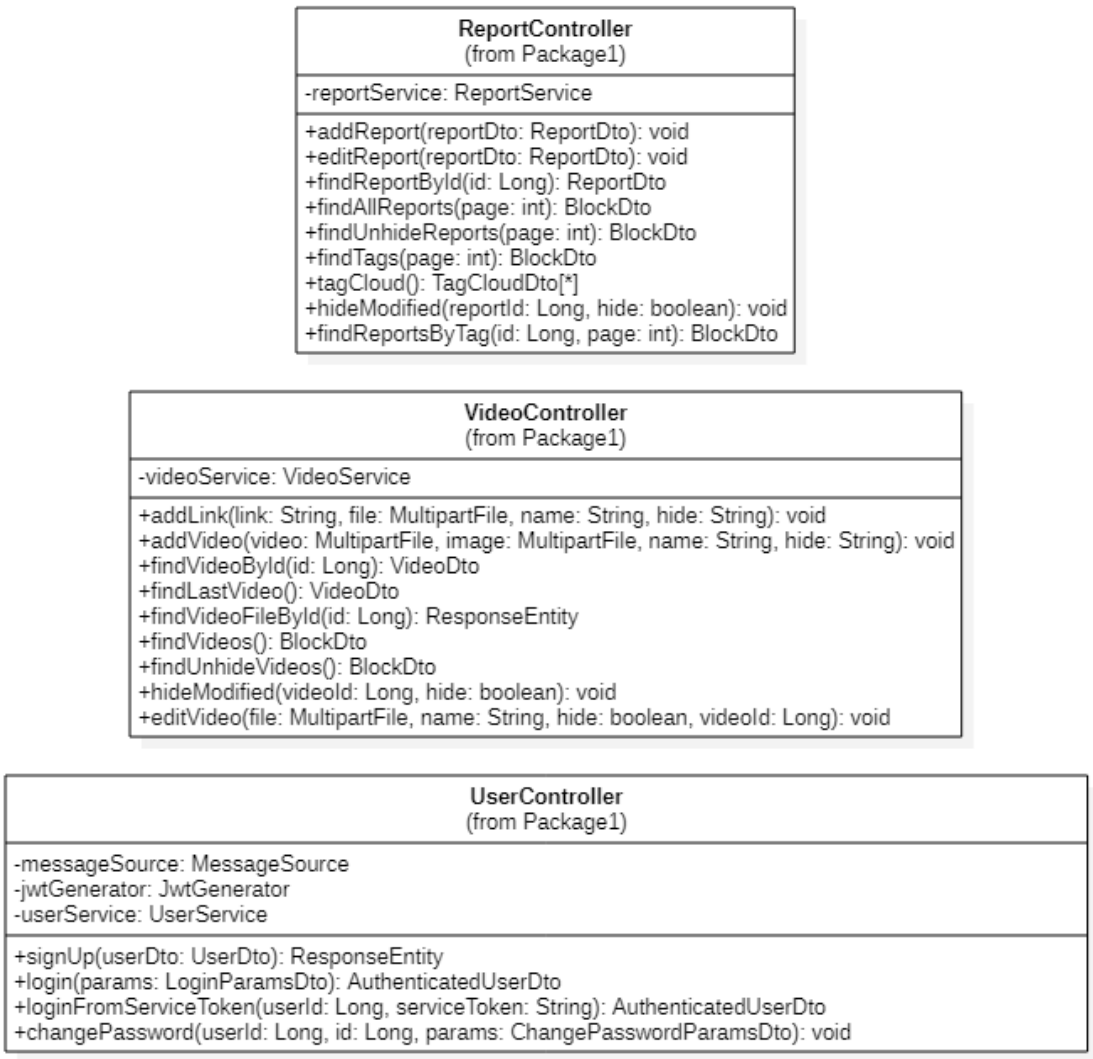

Figura 6.11: Controladores

ReportController

ReportController se encarga de las peticiones relacionadas con las noticias.Su path es *"/report"* y sus funciones realizan una llamada a la función correspondiente del servicio.

- addReport: Recibe un **POST** *"/addReport"* con una noticia y devuelve **201 Created**.
- editReport: Recibe una petición **POST** *"/editReport"* con una noticia y devuelve **204 No Content**.
- findReportById: Recibe un **GET** *"/id"* y devuelve un objeto del tipo **ReportDto**.

- findAllReports: Recibe un **GET** *"/findAllReports"* con el número de página y devuelve un objeto del tipo **BlockDto<ReportDto>**.
- findUnhideReports: Recibe un **GET** *"/findUnhideReports"* con el número de página y devuelve un objeto del tipo **BlockDto<ReportDto>**.
- findTags: Recibe un **GET** *"/findTags"* con el número de página y devuelve un objeto del tipo **BlockDto<TagDto>**.
- tagCloud: Recibe un **GET** *"/tagCloud"* y devuelve un objeto del tipo **List<tagCloud>**.
- hideModified: Recibe un **POST** *"/hideModified"* con un id y un boolean y devuelve **204 No Content**.
- findReportsByTag: Recibe un **GET** *"/findReportsByTag"* con el id de la etiqueta y el número de página y devuelve un objeto del tipo **BlockDto<ReportDto>**.

VideoController

VideoController se encarga de las peticiones relacionadas con los vídeos. Su *path* es *"/video"* y sus funciones realizan una llamada a la función correspondiente del servicio.

- addLink: Recibe un **POST** *"/addVideoLink"* con la información para crear un vídeo del tipo *link* y devuelve **201 Created**.
- addVideo: Recibe un **POST** *"/addVideo"* con un vídeo del tipo archivo y devuelve **201 Created**.
- findVideoById: Recibe un **GET** *"/id"* y devuelve un objeto del tipo **VideoDto**.
- findLastVideo: Recibe un **GET** *"/last"* y devuelve un objeto del tipo **VideoDto**.
- findVideoFileById: Recibe un **GET** *"/findVideoById/id"* y devuelve un objeto del tipo **ResponseEntity<Resource>**.
- findVideos: Recibe un **GET** *"/findVideos"* y devuelve un objeto del tipo **BlockDto<VideoDto>**.
- findUnhideVideos: Recibe un **GET** *"/findUnhideVideos"* y devuelve un objeto del tipo **BlockDto<VideoDto>**.
- hideModified: Recibe un **POST** *"/hideModified"* con un *id* y un *boolean* y devuelve **204 No Content**.
- editVideo: Recibe una petición **POST** *"/editVideo"* con un *id*, un nombre, un *boolean* y una imagen (si la hay) y devuelve **204 No Content**.

UserController

UserController se encarga de las peticiones relacionadas con los usuarios. Su path es *"/user"* y sus funciones realizan una llamada a la función correspondiente del servicio.

- signUp: Recibe un **POST** *"signUp"* con el usuario a crear y devuelve un objeto del tipo **ResponseEntity<AuthenticatedUserDto>**.
- login: Recibe un **GET** *"login"* con los parámetros de inicio de sesión y devuelve un objeto del tipo **AuthenticatedUserDto**.
- loginFromServiceToken: Recibe un **GET** *"loginFromServiceToken"* con el *id* de usuario y su *token* y devuelve un objeto del tipo **AuthenticatedUserDto**
- changePassword: Recibe un **Post** *"/id/changePassword"* con el *id* de usuario y los parámetros de cambio de contraseña y devuelve **204 No Content**. A mayores comprueba si el *id* del usuario que mando la petición es el mismo que el usuario que intenta cambiar la contraseña.

6.5 Capa interfaz

La capa de la interfaz ha sido desarrollada en **JavaScript** y **HTML** utilizando *React* con la librería *Redux*. De esta manera, gestionaremos la plataforma mediante estados, que son un único objeto *JavaScript*, organizado en árbol y que contiene todos los datos que la aplicación va a manejar.

Al usar *NPM* y *React* se han podido usar varias librerías esenciales para poder llevar a cabo los requisitos.

- **react-files:** interfaz para subida de archivos.
- **react-ticker:** para realizar el *scroll* lateral a modo noticiario.
- **react-player:** que ha servido para poder reproducir vídeos tanto de archivos(compatibles con <video>) como de diferentes plataformas (*Youtube, Facebook, SoundCloud, Streamable, Vimeo, Wistia, Twitch, DailyMotion and Vidyard*) utilizando un mismo elemento jugando con sus configuraciones.
- **react-tagcloud:** para realizar la nube de etiquetas.

Para poder visualizar mejor como es el funcionamiento, en vez de diagramas *UML* con todos las mismas funciones, se realizarán unos diagramas con los que se podrá ver el orden de ejecución de las clases y qué hacen.

El esquema seguido para el desarrollo de esta interfaz consta de una clase que almacena las llamadas y redirige las peticiones (listado6.2); una clase que recibe esa redirección (listado 6.3), manda la petición al *backend* y llama a la case de renderizado (esta clase intermedia en ocasiones no existe, debido a que se hace la petición ya en base al *link*, como son por ejemplo los links con un id); y por último una clase que se encarga de renderizar el resultado y de mandar una respuesta si es necesario (listado 6.4). A su vez la clase *render* puede llamar a otras clases *render* en caso de tener varios elementos.

```
1  const Body = ({loggedIn}) => (
2
3      <div className="container">
4          <br/>
5          <AppGlobalComponents/>
6          <Switch>
7              <Route exact path="/home/:id"
       component={withRouter(HomeId)}/>
8              <Route exact path="/home" component={Home}/>
9
10             {loggedIn && <Route exact path="/users/add-link"
       component={AddLink}/>}
11             {loggedIn && <Route exact path="/users/add-video"
       component={AddVideo}/>}
12             {loggedIn && <Route exact path="/users/add-report"
       component={AddReport}/>}
13             {loggedIn && <Route exact path="/users/change-password"
       component={ChangePassword}/>}
14             {loggedIn && <Route exact path="/users/logout"
       component={Logout}/>}
15             {!loggedIn && <Route exact path="/users/login"
       component={Login}/>}
16             {!loggedIn && <Route exact path="/users/signup"
       component={SignUp}/>}
17             <Route exact path="/report"
       component={FindUnhideReports}/>
18             <Route exact path="/tags" component={FindTags}/>
19             <Route exact path="/report/report-details/:id"
       component={withRouter(ReportDetails)}/>
20             <Route exact path="/report/findReportsTag/:id"
       component={withRouter(FindReportsTag)}/>
21             {loggedIn && <Route exact path="/users/modifyNews"
       component={FindAllReports}/>}
22             {loggedIn && <Route exact
       path="/report/editReportDetails/:id"
       component={withRouter(EditReportDetails)}/>}
23             {loggedIn && <Route exact path="/users/modifyVideos"
```

```
    component={FindAllVideos}/>}
24              {loggedIn && <Route exact
    path="/report/editVideoDetails/:id"
    component={withRouter(EditVideoDetails)}/>}
25
26              <Route component={Home}/>
27          </Switch>
28      </div>
29  );
30
31  const mapStateToProps = state => ({
32      loggedIn: users.selectors.isLoggedIn(state)
33  });
34
35  export default withRouter(connect(mapStateToProps)(Body));
```

Listado 6.2: Ejemplo body

En el listado 6.2 podemos ver un ejemplo de como es la clase que recibe las llamadas por *URL*.

```
1   class Home extends React.Component {
2
3
4       componentDidMount() {
5
6           this.props.dispatch(actionsVideoLinks.findLastVideoLink());
        //Pedimos el último video visible
7
8           this.props.dispatch(actionsVideoLinks.viewLinksUser());
        //Pedimos la lista de videos
9           this.props.dispatch(actionsReports.viewUnhideReports(0));
        //Pedimos la lista de noticias para hacer el carrusel
10
11      }
12
13      render() {
14          return(
15              <HomeResult/>  //Llamada a la clase render
16          );
17      }
18
19  }
20
21  export default connect()(Home);
```

Listado 6.3: Ejemplo clase intermediaria

En el listado 6.3 vemos el ejemplo de como se llaman a las funciones que nos brindan la información de la pantalla de inicio.

```
const HomeResult = ({userVideoLinkSearch, userReportSearch, tags,
    videoLink}) => {
    return (

        <div className="container">
                <div className="row section1">
                    <div className="col-xl-8 col-lg-8 col-md-12">
                        <Player videoLink={videoLink}/>
                    </div>
                    <div className="col-xl-4 col-lg-4 col-md-12">
                        <div className="lista">
                            <VideoLinkResult
    userVideoLinkSearch={userVideoLinkSearch}/>
                        </div>
                    </div>

                </div>


            <div className="row align-items-end">
                <div className="col-md-12 scrollReport">
                    <HomeReportResult
    reports={userReportSearch.result.items}/>
                </div>

            </div>
        </div>

    );

}

const mapStateToProps = (state) => ({
    userReportSearch: selectorReports.getUserReportSearch(state),
```

Listado 6.4: Ejemplo clase render

En el listado 6.2 se puede observar como ya incluye código *HTML* para dividir las secciones de la página, y cómo llama al resto de clases para que renderizen la información que les corresponde.

A continuación podemos observar como es el diagrama de flujo de las funciones del usuario normal (figura 6.12) y las del administrador (figura 6.13.)

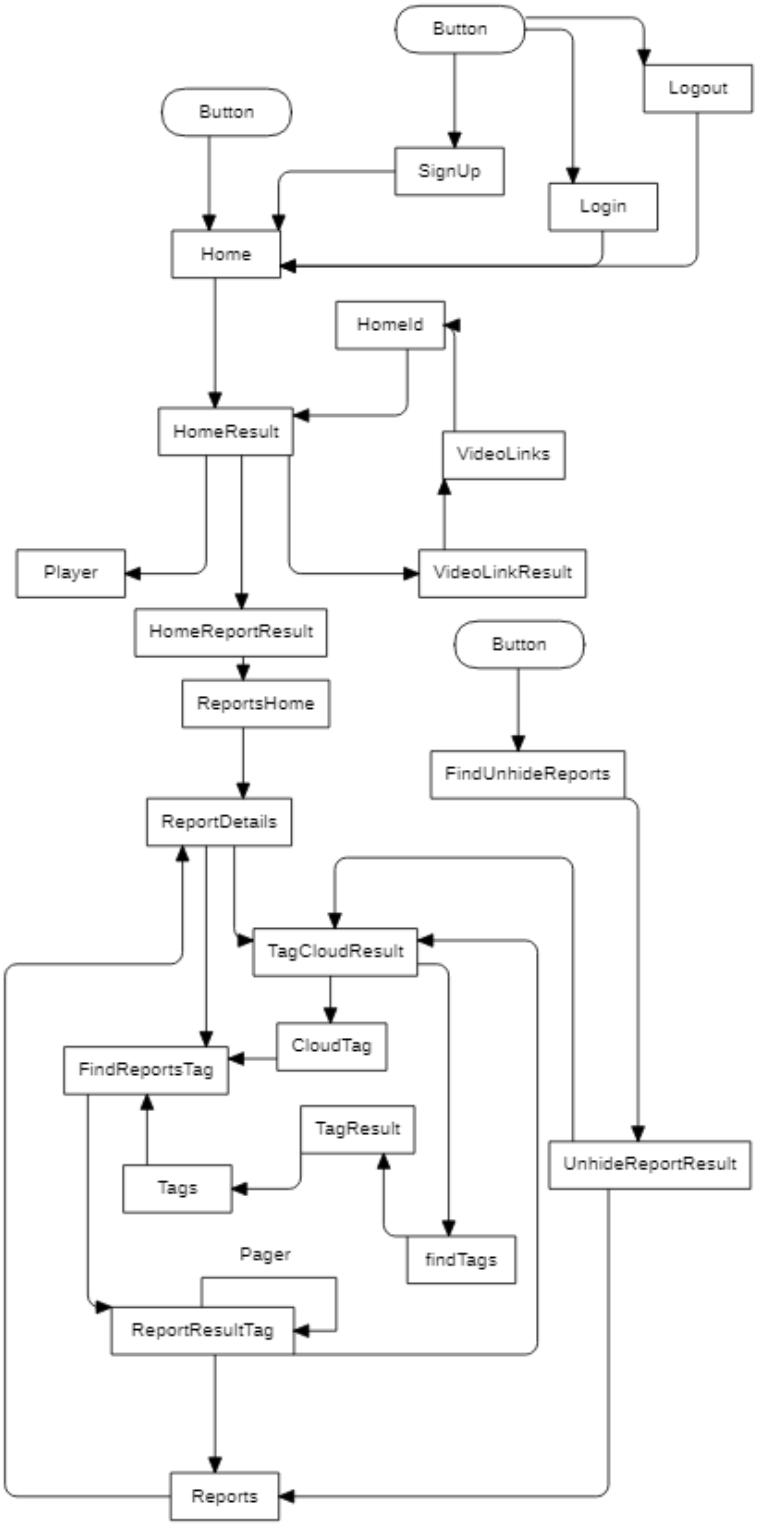

Figura 6.12: Diagrama flujo usuario

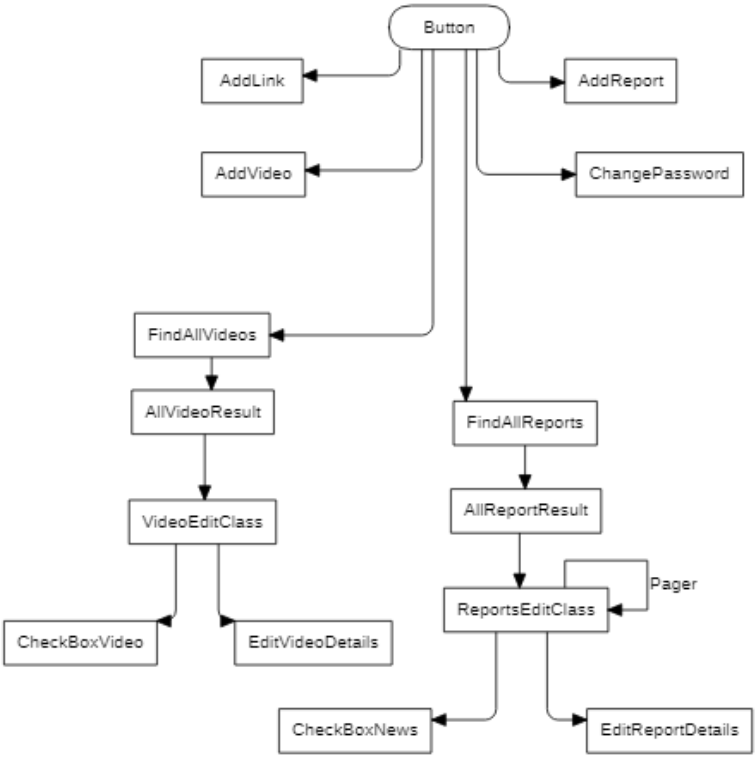

Figura 6.13: Diagrama flujo administrador

La interfaz se ha realizado en inglés, castellano y gallego. Éstos se seleccionan a partir del idioma del navegador y se han añadido mediante **i18n**, que es dónde se almacenan los ficheros con los idiomas.

Por último, con ayuda de **CSS** se ha realizado un diseño tanto para modo claro como para modo oscuro, de manera que sea fácil de visualizar en cualquier momento.

Se puede ver el resultado de la plataforma en el apéndice A.

Capítulo 7

Pruebas

7.1 Introducción

Para el primer *Sprint* se han seguido las tres fases, es decir, primero se diseñó una plataforma con sólo un modelo, en el cual se hicieron pruebas unitarias. Una vez realizado, se diseñaron unos controladores y se probaron mediante las pruebas de integración. Por último se diseñó la parte web y se hicieron las pruebas del sistema.

Para los posteriores sprints se conservaron y modificaron las pruebas unitarias y se hicieron varias de integración, pero nos centramos en las pruebas del sistema, debido a que ya existía una plataforma estable en la cual interactuar.

7.2 Primera fase (unitarias)

Sirven para comprobar el correcto funcionamiento de una unidad de código. En el proyecto su principal función fue comprobar el funcionamiento del acceso a la base de datos y los casos de uso básicos. Además comprobamos el funcionamiento de las entidades y sus relaciones básicas. Para ello se utilizo la librería **JUnit** integrada en Eclipse. Además puede hacerse de manera integrada con **Maven**. Con **JUnit** conseguimos aislar cada parte del programa de forma que cada prueba se ejecute de manera independiente, por tanto no propagamos errores entre cada prueba.

7.3 Segunda fase (integración)

Las pruebas de integración sirven para comprobar que todos los componentes funcionen correctamente interactuando entre si. Estas pruebas nos ayudan a comprobar que hay una correcta conexión entre los componentes así como posibles fallos en el código de los controladores de la aplicación.

Estas pruebas se han realizado una vez las pruebas unitarias de la primera interacción funcionan de manera correcta y mostraban los resultados previstos.

Como se han diseñado unos controladores usando *REST* se ha utilizado **Postman** como aplicación de pruebas, haciendo peticiones como por ejemplo pedir la lista de noticias, la lista de vídeos, o probar las adiciones de elementos.

A parte de **Postman** también se han realizado pruebas desde el propio cliente haciendo las peticiones y analizando las respuestas.

7.4 Tercera fase(sistema)

Una vez realizadas las pruebas de integración y sabiendo que las peticiones van a responderse con lo deseado, se ha procedido a la elaboración del *FrontEnd* con su respectiva interfaz gráfica.

Aún sin diseño definido se han ido haciendo pruebas de cada acción según se han ido implementado las opciones en la interfaz.

7.4.1 Pruebas funcionales

Una vez realizada toda la interfaz y aplicado un diseño que no sea el del entorno de pruebas, se ha pasado a hacer una serie de pruebas funcionales que no sólo comprobaban el funcionamiento de las acciones, sino el comportamiento de la página a la hora de realizarlas. Esto puede ser por ejemplo: al añadir una noticia, el sistema informa que la noticia está añadida en una pestaña emergente, que indica el nombre de la noticia, y una vez se cierra la ventana retorna al inicio.

Estas pruebas se dividen en las siguientes partes:

Pruebas de Administrador.

Estas pruebas son las que comprueban el funcionamiento de un administrador de la plataforma.

Nº	Pruebas de Administrador.	Resultado	Check
1	Registro con usuario duplicado.	Indica que el usuario ya existe.	ok
2	Registro con contraseña errónea (mal escrita la segunda vez).	Avisa de que la contraseña no corresponde.	ok
3	Registro correcto.	Si todo va bien, se inicia sesión con dicho registro y pasa a la página de inicio.	ok
4	Inicio de sesión con contraseña incorrecta.	Indica de usuario o contraseña erróneos.	ok
5	Inicio de sesión con usuario no existente.	Indica de usuario o contraseña erróneos.	ok
6	Inicio de sesión correcto.	Se loquea y pasa a la página de inicio.	ok
7	Cerrar sesión.	Cierra sesión y se pasa a la página de inicio.	ok
8	Añadir vídeo por link con link incorrecto.	Indica que el link es incorrecto.	ok
9	Añadir vídeo por link sin nombre.	Indica que es un campo requerido.	ok
10	Añadir vídeo por link sin miniatura.	Indica que es necesaria una miniatura.	ok
11	Añade vídeo por link bien, sin ser hide.	Añade el vídeo de manera normal.	ok
12	Añadir vídeo por link bien, siendo hide.	Añade el vídeo en oculto.	ok
13	Añadir vídeo por link pero con un link igual a uno existente.	Avisa de la existencia del vídeo en la web.	ok
14	Añadir un vídeo con un nombre existente.	Avisa de la existencia del vídeo en la web.	ok
15	Añadir vídeo sin nombre.	Indica que es un campo requerido.	ok
16	Añadir vídeo pero sin dar el archivo.	Avisa de que hay que añadir el vídeo.	ok
17	Añadir vídeo sin la foto.	Avisa de que hay que añadir una imagen.	ok
18	Añadir el vídeo bien sin ocultar.	Añade el vídeo de manera normal.	ok
19	Añadir el vídeo bien pero oculto.	Añade el vídeo en oculto.	ok

20	Añadir un vídeo cuyo nombre de archivo coincida con uno de la base de datos.	Avisa de la existencia del vídeo en la web.	ok
21	Añadir un vídeo con un nombre existente.	Avisa de la existencia del vídeo en la web.	ok
22	Añadir noticia sin titular.	Indica que es un campo requerido.	ok
23	Añadir noticia sin cuerpo.	Indica que es un campo requerido.	ok
24	Añadir noticia sin etiquetas.	Añade la noticia sin etiquetas.	ok
25	Añadir noticia con una etiqueta en el primer campo.	Añade la noticia con una etiqueta.	ok
26	Añadir noticia con una etiqueta en el segundo campo.	Añade la noticia con una etiqueta.	ok
27	Añadir noticia con una etiqueta en el tercer campo.	Añade la noticia con una etiqueta.	ok
28	Añadir noticia con dos etiquetas pero el primer campo vacío.	Añade una noticia con dos etiquetas.	ok
29	Añadir noticia con dos etiquetas pero el segundo campo vacío.	Añade una noticia con dos etiquetas.	ok
30	Añadir noticia con dos etiquetas pero el tercer campo vacío.	Añade una noticia con dos etiquetas.	ok
31	Añadir noticia con tres etiquetas.	Añade una noticia con tres etiquetas.	ok
32	Añadir una noticia oculta.	Añade una noticia oculta.	ok
33	Añadir noticia con titular repetido.	Indica que la noticia ya existe.	ok
34	Ocultar una noticia desde el menú de modificar noticias.	Oculta la noticia.	ok
35	Hacer una noticia visible desde el menú de modificar noticias.	Hace que la noticia sea visible.	ok
36	Modificar el titular de una noticia.	Modifica el titular.	ok
37	Modificar cuerpo de una noticia.	Se modifica el cuerpo de la noticia.	ok
38	Modificar una etiqueta de una noticia.	Se modifican tal y como funciona el añadir noticia.	ok
39	Ocultar una noticia.	Se oculta la noticia.	ok
40	Mostrar una noticia.	Hace visible la noticia.	ok
41	Dejar el nombre en blanco al modificar.	Indica campo requerido.	ok
42	Dejar el cuerpo en blanco al modificar.	Indica campo requerido.	ok
43	Añadir noticia con titular repetido.	Indica que la noticia ya existe.	ok

44	Cambiar contraseña con la contraseña anterior mal.	Indica que la contraseña es erronea.	ok
45	Cambiar contraseña con mala confirmación de contraseña.	Indica que las contraseñas no coinciden.	ok
46	Cambiar la contraseña de manera correcta.	Cambia la contraseña.	ok
47	Ocultar un vídeo desde el menú de modificar vídeos.	Oculta el vídeo de la pantalla principal.	ok
48	Hacer un vídeo visible desde el menú de modificar vídeos.	El vídeo se hace visible.	ok
49	Modificar nombre de un vídeo.	Se modifica el nombre del vídeo.	ok
50	Modificar la miniatura de un vídeo.	Se modifica la miniatura del vídeo.	ok
51	Dejar el campo de nombre en blanco al modificar.	Indica campo requerido.	ok
52	Dejar el campo de foto en blanco al modificar.	No es campo obligatorio, por tanto deja la anterior.	ok
53	Ocultar un vídeo.	Oculta el vídeo de la pantalla principal.	ok
54	Mostrar un vídeo.	El vídeo se hace visible.	ok
55	Ponerle a un vídeo un nombre de otro vídeo ya existente.	Indica que ya existe un vídeo con ese nombre.	ok

Pruebas de Cliente.

Estas pruebas son las que comprueban el funcionamiento de un usuario normal que visita la plataforma.

Nº	**Pruebas de Cliente.**	**Resultado.**	**Check**
1	Visualización de la página principal.	Correcto.	ok
2	Acceso desde el botón ”Inicio”.	Accede a la página principal.	ok
3	Acceso desde el logo o el nombre de la compañía.	Accede a la página principal.	ok
4	Controles del vídeo de la página principal (origen link).	Son accesibles y funcionales.	ok
5	Controles del vídeo de la página principal (origen archivo).	Son accesibles y funcionales.	ok
6	Desplazamiento por la lista de vídeos.	Se visualizan las miniaturas y los vídeos de manera correcta.	ok

7	Acceso al vídeo deseado de la lista de vídeos.	Se accede correctamente sea del origen que sea.	ok
8	Visualización del scroll de noticias.	Se visualiza sin problemas y en constante bucle de recientes.	ok
9	Acceso a una noticia del scroll.	Se accede sin problemas.	ok
10	Visualización de la página de noticias.	Correcto.	ok
11	Acceso desde el botón "Noticias".	Accede a la página de noticias.	ok
12	Navegación hacia atrás desde la primera página de noticias.	Bloqueado.	ok
13	Navegación hacia delante (Noticias).	Avanza sin problemas.	ok
14	Navegación hacia atrás (Noticias).	Retrocede sin problemas.	ok
15	Navegación hacia delante desde la última página de noticias.	Bloqueado.	ok
16	Acceso a una noticia.	Se accede sin problemas.	ok
17	Acceso a las etiquetas de la nube de etiquetas (Noticias).	Se accede sin problemas.	ok
18	Acceso a la visualización de todas las etiquetas (botón "+more etiquetas") (Noticias).	Se accede sin problemas.	ok
19	Navegación hacia atrás desde la primera página de etiquetas.	Bloqueado.	ok
20	Navegación hacia delante (Etiquetas).	Avanza sin problemas.	ok
21	Navegación hacia atrás (Etiquetas).	Retrocede sin problemas.	ok
22	Navegación hacia delante desde la última página de etiquetas.	Bloqueado.	ok
23	Acceso a una etiqueta.	Se accede sin problemas.	ok
24	Visualización de la noticia	Correcto.	ok
25	Acceso a las etiquetas de la nube de etiquetas (Noticia).	Se accede sin problemas.	ok
26	Acceso a la visualización de todas las etiquetas (botón "+more etiquetas") (Noticia).	Se accede sin problemas.	ok
27	Diferenciación de titular, cuerpo y etiquetas.	Se diferencian claramente los 3 campos por separado.	ok
28	Acceso a las etiquetas de la noticia en caso de tenerlas.	Se accede sin problemas.	ok

Pruebas de Navegador.

Estas pruebas son correspondientes a opciones que tienen que ver con el navegador o con el propio diseño de la plataforma.

Nº	Pruebas de Navegador.	Resultado.	Check
1	Funcionalidad menú retráctil.	Se adapta sin problemas a diferentes tamaños.	ok
2	Acceso a los botones del menú retráctil.	Mantienen su funcionalidad en diferentes tamaños.	ok
3	Cambio entre modo claro y modo oscuro.	Funciona.	ok
4	Visibilidad de todos los elementos en modo claro.	Se ve todo de manera clara.	ok
5	Visibilidad de todos los elementos en modo oscuro.	Se ve todo de manera clara. (Pero oscuro).	ok
6	Funcionalidad del botón de refresco.	Funciona de manera correcta.	ok
7	Funcionalidad del botón ”Ir a página siguiente”.	Funciona de manera correcta.	ok
8	Funcionalidad del botón ”Ir a página anterior”.	Funciona de manera correcta.	ok
9	Correcta visualización del modo responsive.	Se adapta sin problemas cualquiera de las pantallas.	ok

Capítulo 8

Planificación

Para la planificación, como hemos visto en el capítulo 4 se han utilizado **Sprints** para desarrollar la aplicación. En la figura 8.1 se aprecia lo que vendría a ser la primera iteración. En ella se instaló *Redmine* para sacar la planificación, y se instaló el entorno añadiendo la funcionalidad de los usuario y la inserción de enlaces.

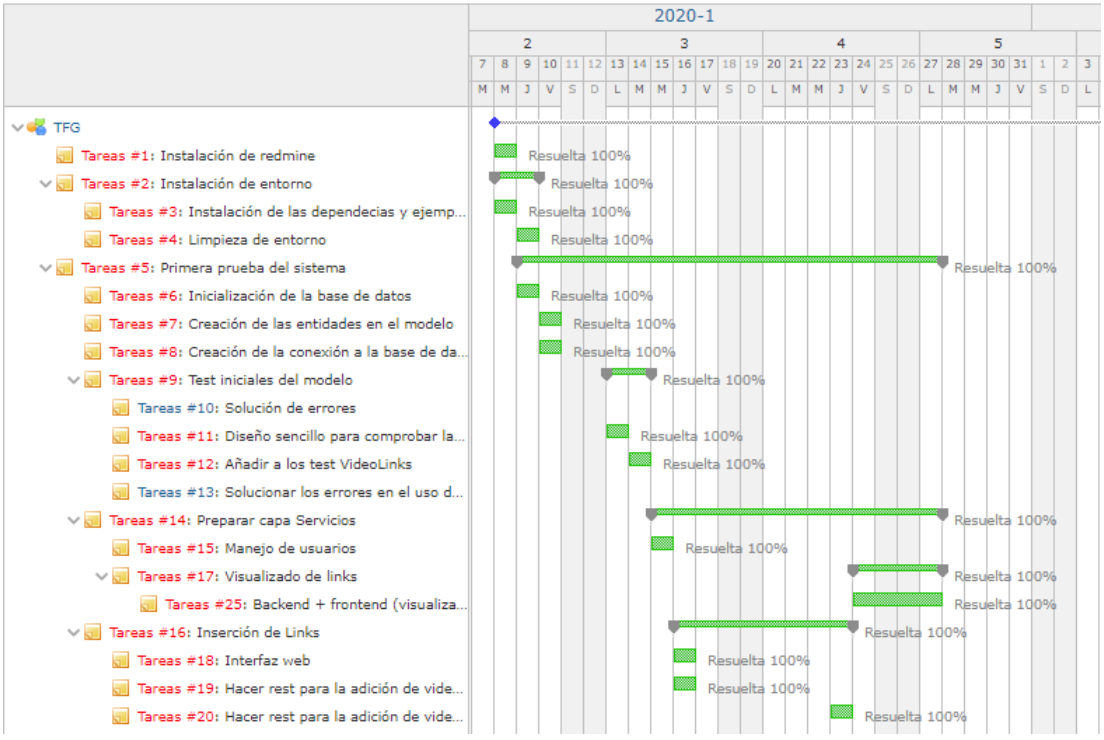

Figura 8.1: Planificación 1

En la figura 8.2 tenemos la segunda iteración que es la adición de noticias, la tercera iteración donde se añadieron las etiquetas junto con la integración de *youtube* y otras plataformas, y el principio de la quinta donde se empieza a elaborar la interfaz.

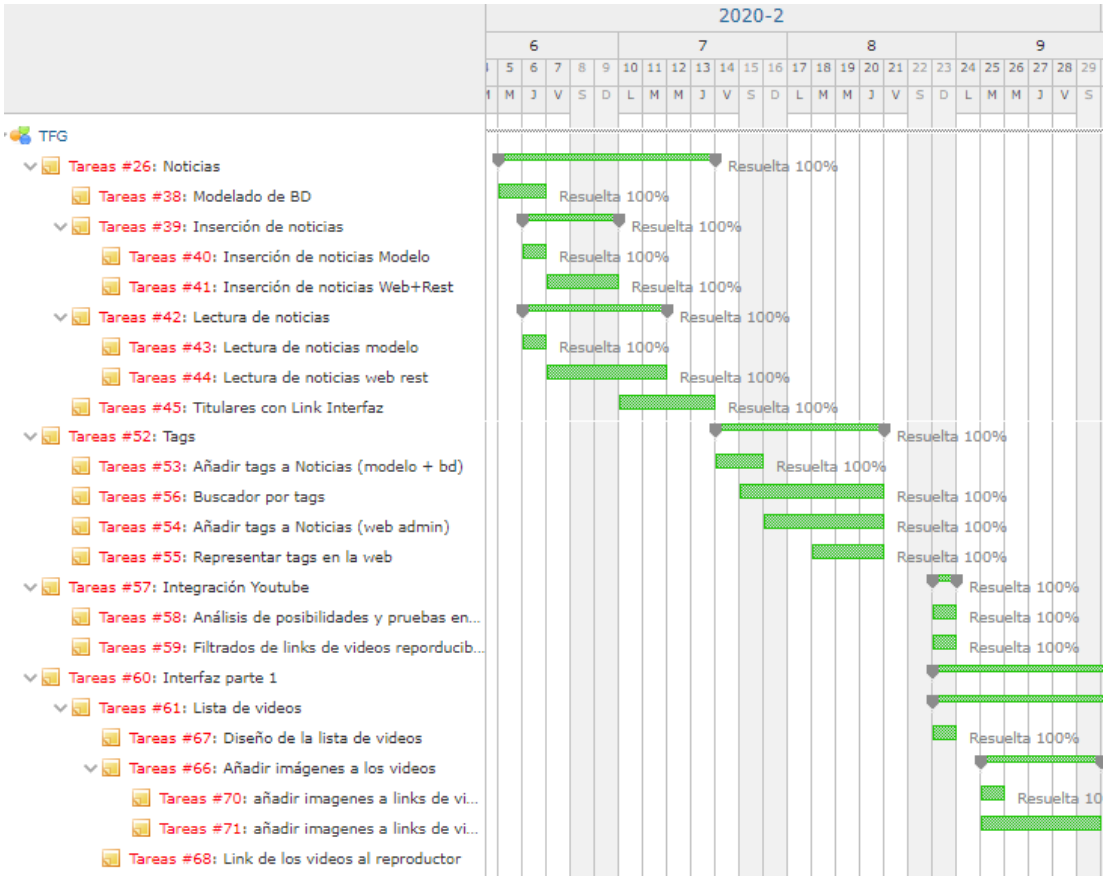

Figura 8.2: Planificación 2

En la 8.3 se visualiza la quinta iteración completa y parte de la textualización de la plataforma, que se fue llevando a cabo en paralelo con las siguientes.

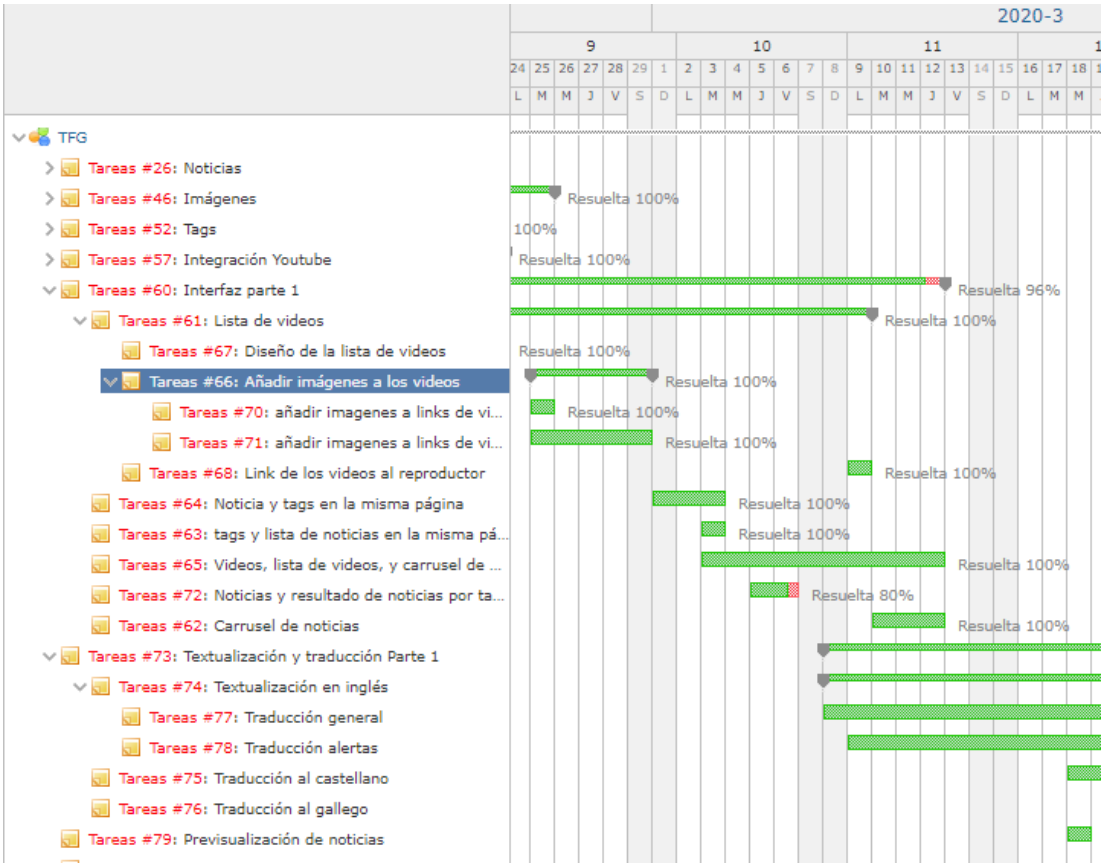

Figura 8.3: Planificación 3

En la 8.4 se observa la sexta iteración, que consta de la ordenación de noticias, la posibilidad de ocultarlas, y la posibilidad de modificarlas. También se ve el comienzo de la séptima, en la cual se añadía la posibilidad de cargar vídeos desde un fichero.

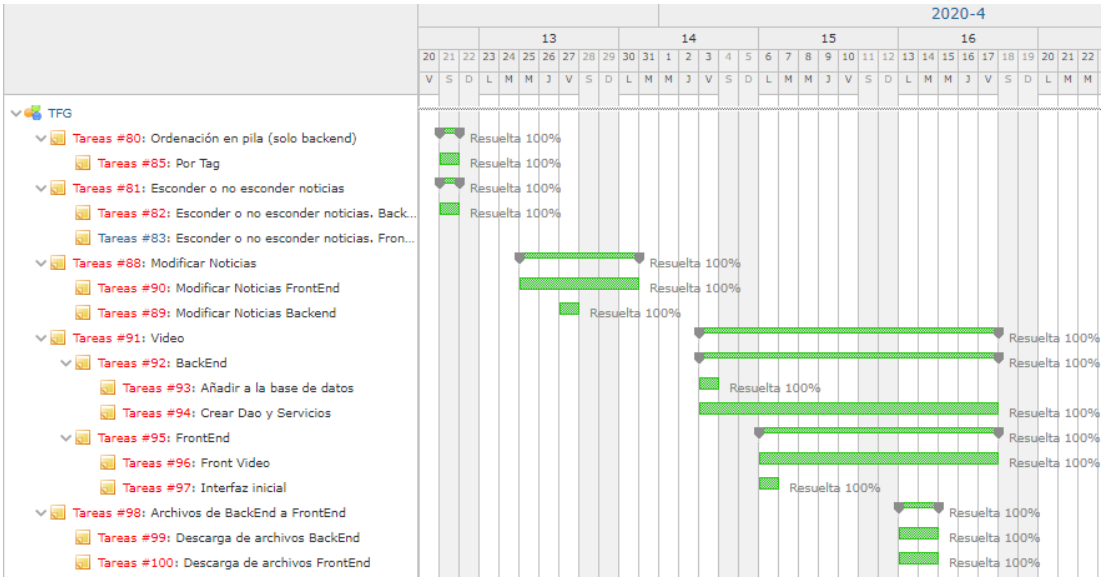

Figura 8.4: Planificación 4

En la 8.5 se aprecia el fin de la séptima iteración y ya el comienzo a escribir de manera más precisa la memoria a partir de las notas tomadas.

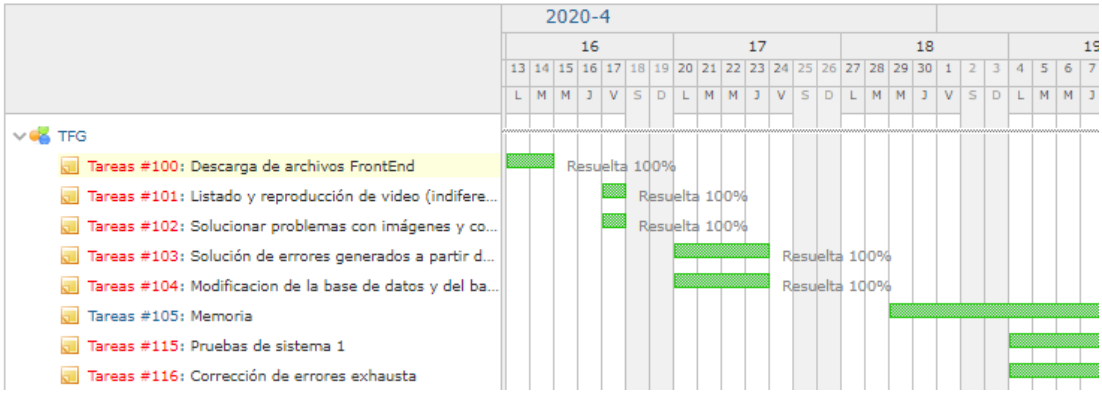

Figura 8.5: Planificación 5

En las figuras 8.6 y 8.7 se aprecia ya el tramo final, al que no le podríamos llamar *Sprint* en si, pero lo podríamos etiquetar como la octava iteración. En ella se lleva a cabo la escritura de la memoria, y se realizan las pruebas de sistema a fondo, junto con el cambio de diseño de la plataforma y la adición del modo noche.

Figura 8.6: Planificación 6

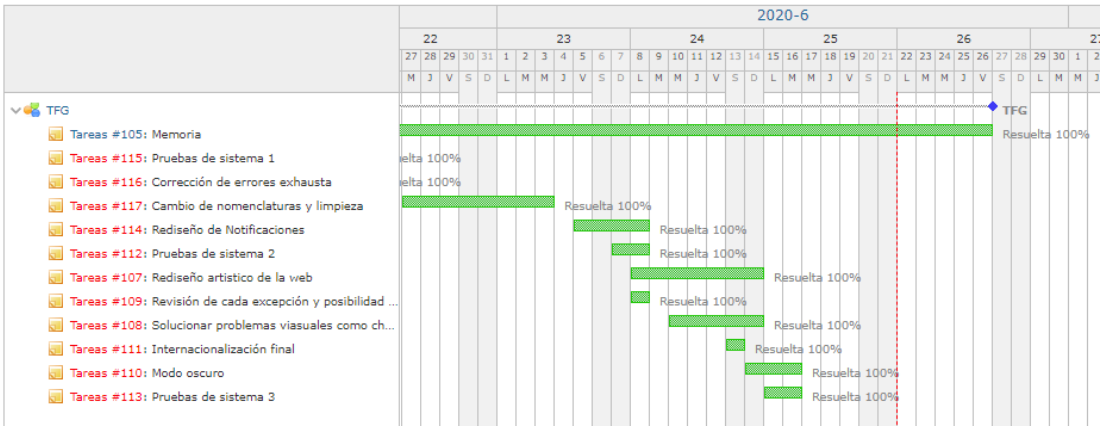

Figura 8.7: Planificación 7

Como se puede ver a lo largo de las imágenes de la planificación, algunas tareas se ve excesivamente largas, otras se separan en el tiempo, etc. Esto se podría llegar a caracterizar como normal, pero se podría haber llevado a cabo de mejor manera si tuviéramos datos previos en los cuales basar el proyecto. Debido a que este es el primero, este recurso no estaba disponible, debido a que es un proyecto individual y no todo el mundo trabaja de la misma manera. Sin embargo, sí se han podido tomar referencias de otros similares. De no haber sido así, probablemente no habría salido en el tiempo estimado.

Capítulo 9

Conclusiones

En este apartado se presentará la situación final del trabajo, las lecciones aprendidas, las relaciones con las competencias de la titulación y de la mención así como las posibles líneas futuras.

En el día que estoy escribiendo esto, el proyecto ha finalizado, todos los objetivos y todas las ideas de mejora que han surgido a lo largo de éste han sido llevadas a cabo de manera satisfactoria. El resultado ha sido una plataforma mejor de lo planeada en un principio, y con una personalización del contenido total.

El requisito de los vídeos era el más complejo a simple vista, y por eso se ha dejado para el final, y ha funcionado. La motivación de ver que el proyecto va cogiendo forma ha sido una clave para querer siempre hacer algo más y nunca dejar de pensar en que mejoras se podían realizar dentro de los propios requisitos preestablecidos.

La planificación realizada se ha llevado a cabo de la mejor manera posible y readaptandola según iban surgiendo los problemas o contratiempos.

Éste es un pequeño resumen, de lo que he realizado en el proyecto:

- He planificado un proyecto y he llevado a cabo dicha planificación.
- He desarrollado una aplicación de principio a fin basándome en el patrón *MVC*.
- He diseñado una plataforma de cliente de cero.
- He programado en *Java*, *SQL*, *JavaScript*, *HTML* y *CSS*.
- He conseguido una plataforma fácil de usar y de mantener.
- He investigado las mejores opciones que tenía para levar a cabo la integración de los requisitos.
- He aprendido tecnologías nuevas y afianzado las ya aprendidas.

9.1 Lecciones aprendidas y relaciones con las competencias

El desarrollo de este proyecto me ha hecho aprender o mejorar el dominio de lo siguiente:

- A planificar el proyecto entero desde un principio. Para realizar esta tarea ha sido de ayuda haber cursado la asignatura de Gestión de Proyectos donde se aprende lo necesario para poder empezar.
- Afianzar los conocimientos sobre la creación de las bases de datos. Utilizando los conocimientos de *Bases de Datos* y de *Administración de Bases de Datos* (especialidad TI).
- Mejorar en el diseño de un modelo basado en java y una conexión a una base de datos. Usando como base los conocimientos de *Internet y Sistemas Distribuidos* y *Programación Avanzada* (especialidad software), aunque esta última no la haya cursado, he tenido acceso a las diapositivas.
- Descubrir más a fondo el mundo *JavaScript* con *React*, siendo esta una tecnología completamente nueva para mí. Esto tiene relación con las asignaturas citadas en la anterior lección y con la asignatura de *Programación Integrativa* (especialidad TI).
- Aprender a utilizar de manera profesional *HTML* y *CSS*, y su integración con *JavaScript* en una plataforma donde la apariencia, las relaciones de tamaños, el diseño responsive y la visibilidad son importantes. Estos temas se trataron por encima en asignaturas como *Interfaz Persona Maquina* o *Programación Integrativa*.
- Desarrollo de *UML* y diagramas de un caso real utilizando los conocimientos de *Proceso de Software* y *Diseño de Software*.

Y, por supuesto, la base adquirida del resto de las asignaturas, sin las cuales, no se podría haber llevado a cabo.

9.2 Líneas Futuras

Con todos los objetivos cumplidos, aún existe la posibilidad de mejora. Independientemente del presupuesto o de la cantidad de desarrolladores, estos podrían ser buenos puntos a llevar cabo para alcanzar una plataforma que no tuviera nada que envidiar:

- Mayor seguridad en la autentificación de administradores.
- Mayor flexibilidad y control de las entidades desde la propia interfaz web.

- Adición de un panel de personalización de la web en el menú de administrador. En la que pueda elegir colores, logo corporativo, etc.
- Modelos precargados de tipos de noticias y botones para inserciones de elementos. De manera que sea aún más sencillo para el redactor.
- Posibilidad de sponsor en las barras laterales.
- Portal de peticiones a la empresa para resolver dudas.
- FAQ de la empresa editable desde el panel del *Administrador*.
- Internacionalización a más idiomas a parte de inglés, gallego y castellano.

Apéndices

Apéndice A

Muestra de la plataforma

A continuación se muestran unas imágenes de como quedaría un ejemplo. Se ha utilizado el juego *Cyberpunk 2077* para rellenar la base de datos simulando así una web para dicho videojuego.

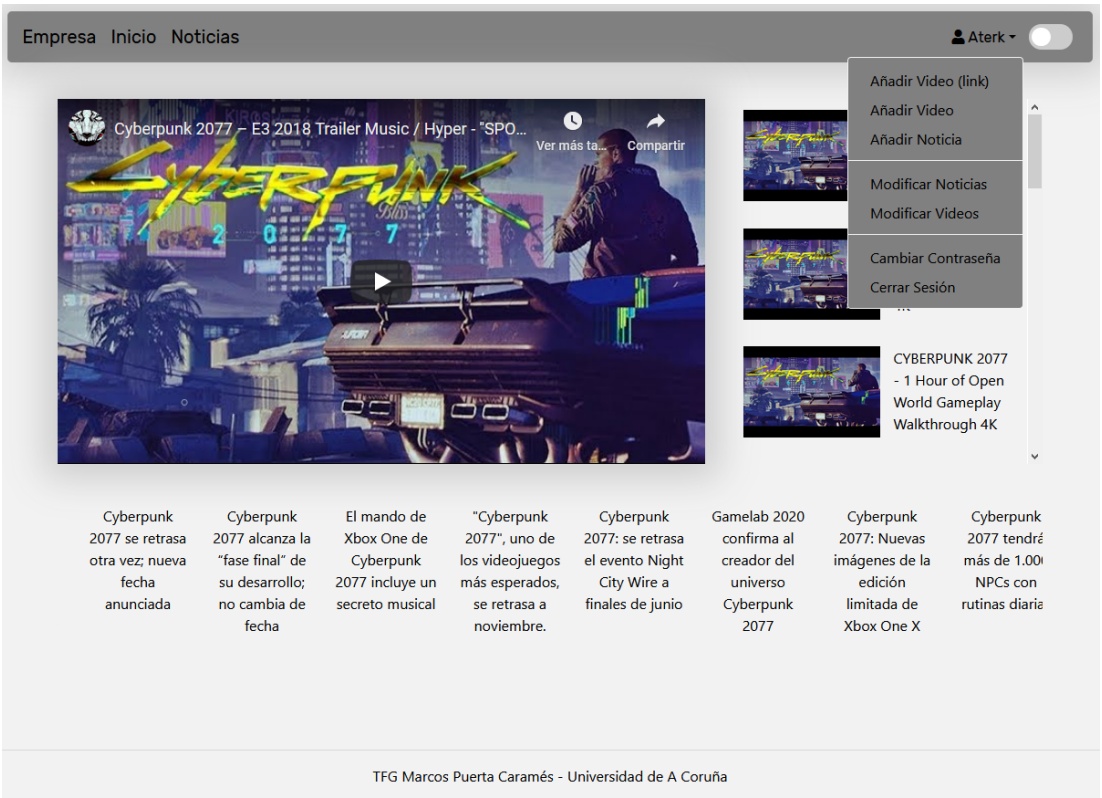

Figura A.1: Pantalla de Inicio

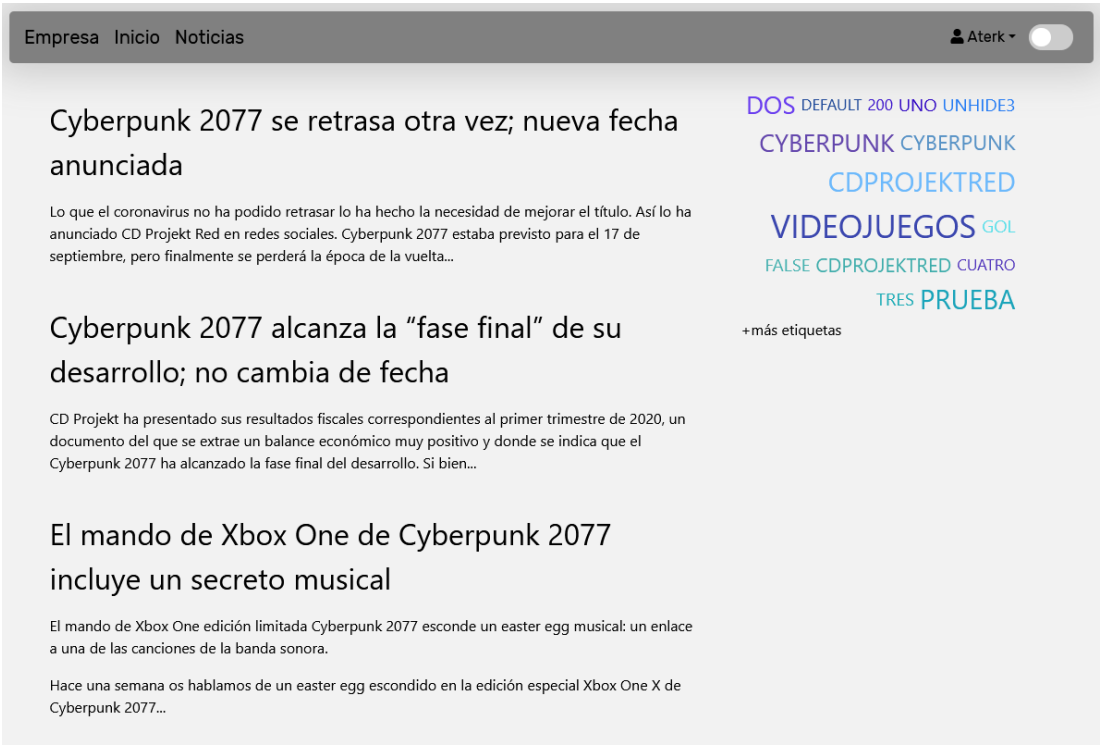

Figura A.2: Pantalla de Noticias

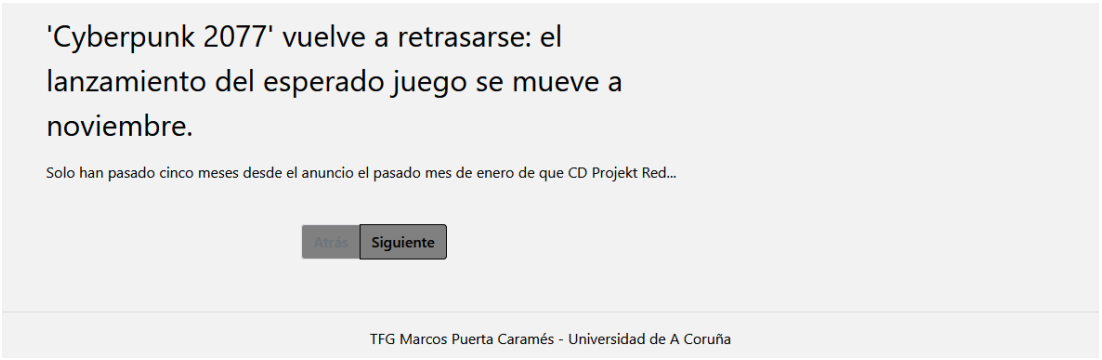

Figura A.3: Pantalla de Noticias 2

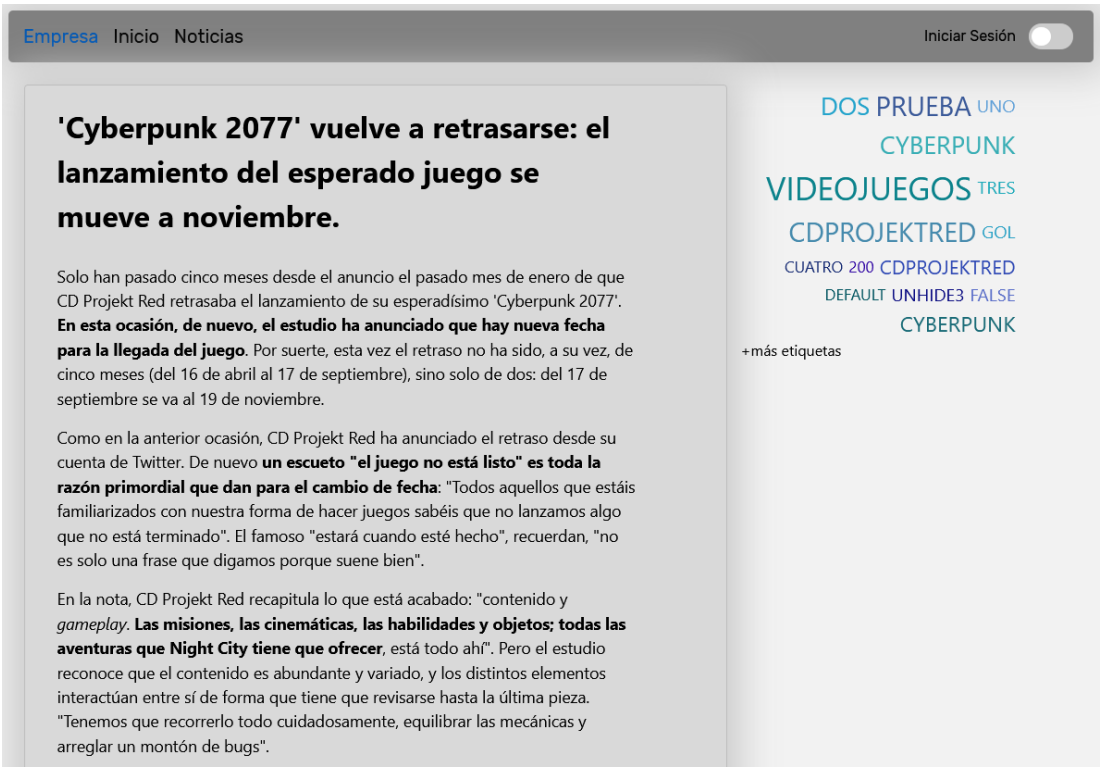

Figura A.4: Pantalla una Noticia

Figura A.5: Pantalla de Tags

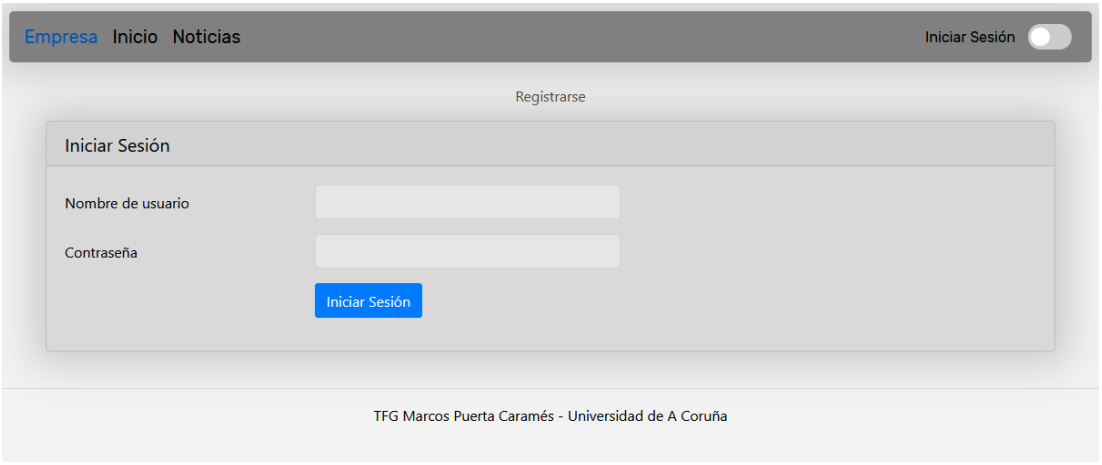

Figura A.6: Pantalla de Inicio de Sesión

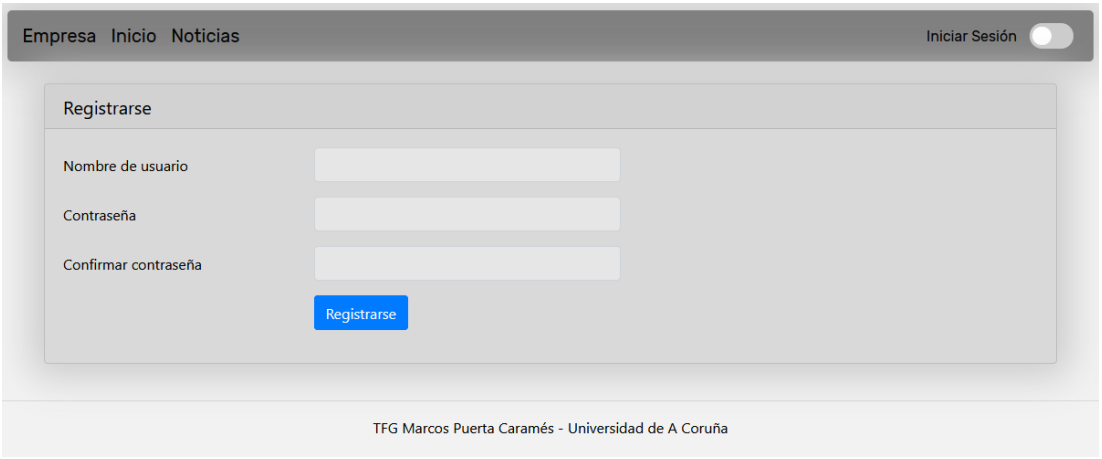

Figura A.7: Pantalla de Registro

Figura A.8: Pantalla de cambio de contraseña

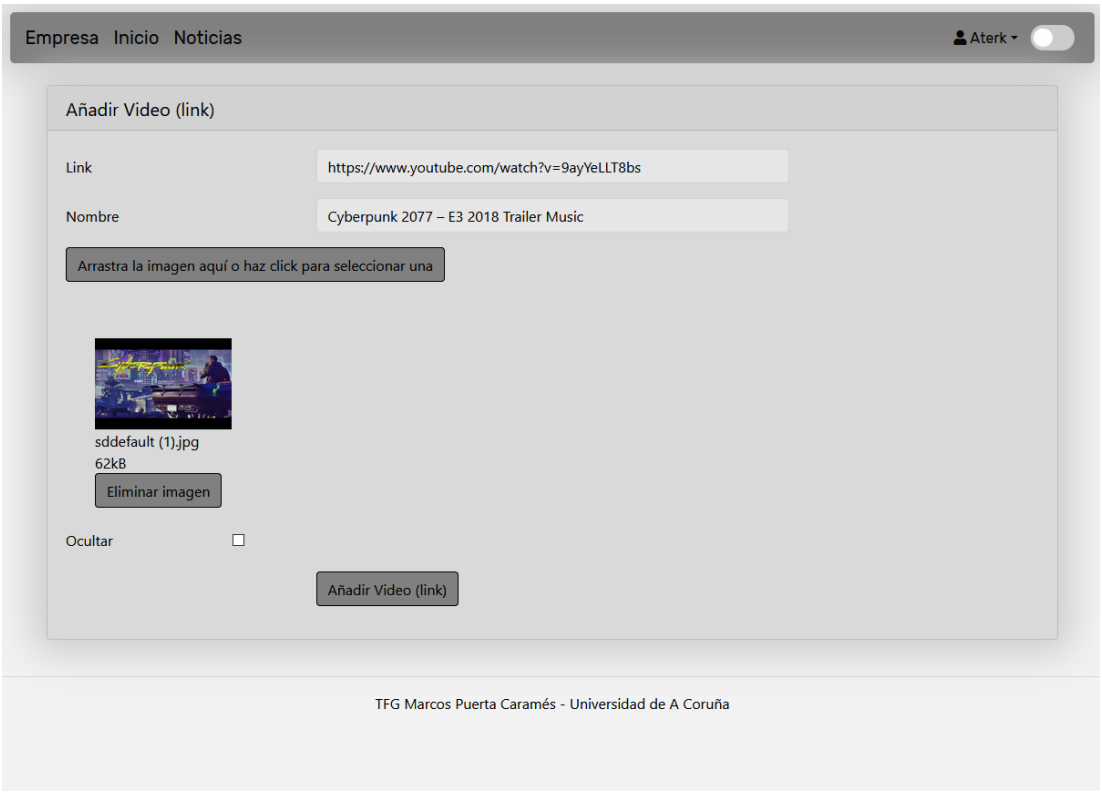

Figura A.9: Pantalla de añadir vídeo por link

Figura A.10: Pantalla de añadir vídeo por archivo

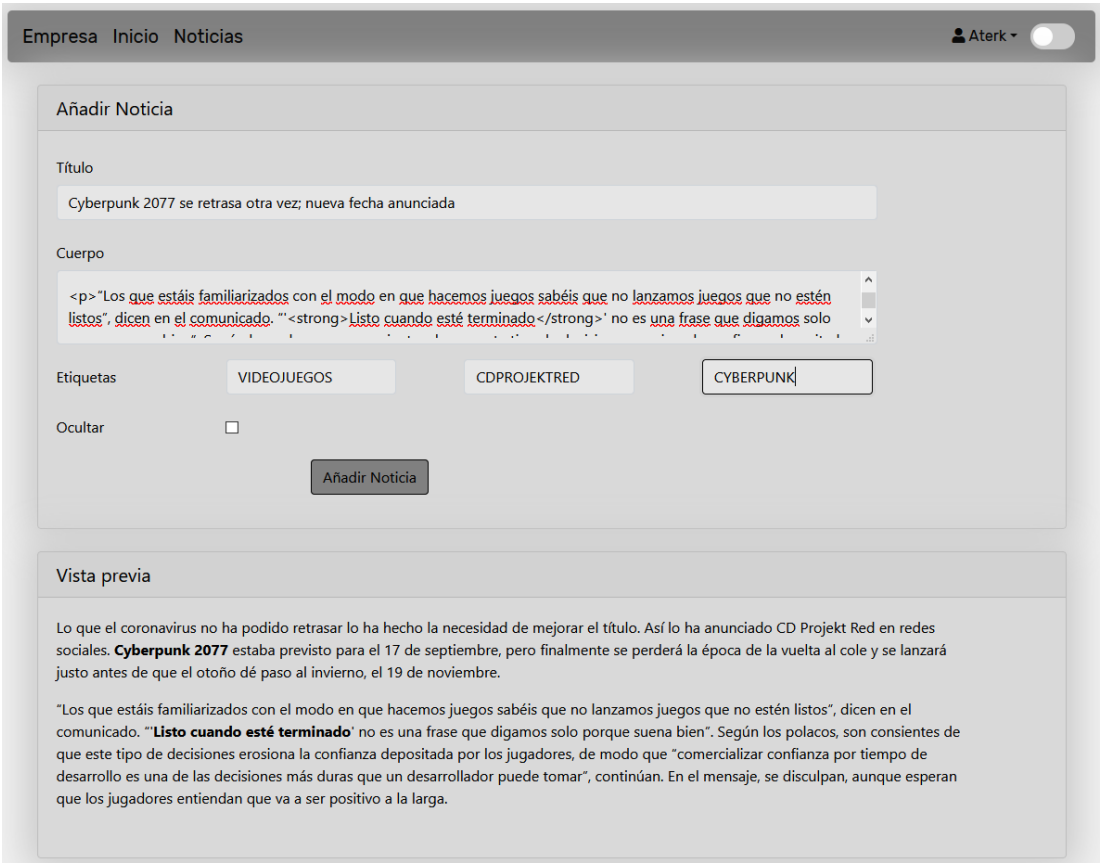

Figura A.11: Pantalla de añadir noticia

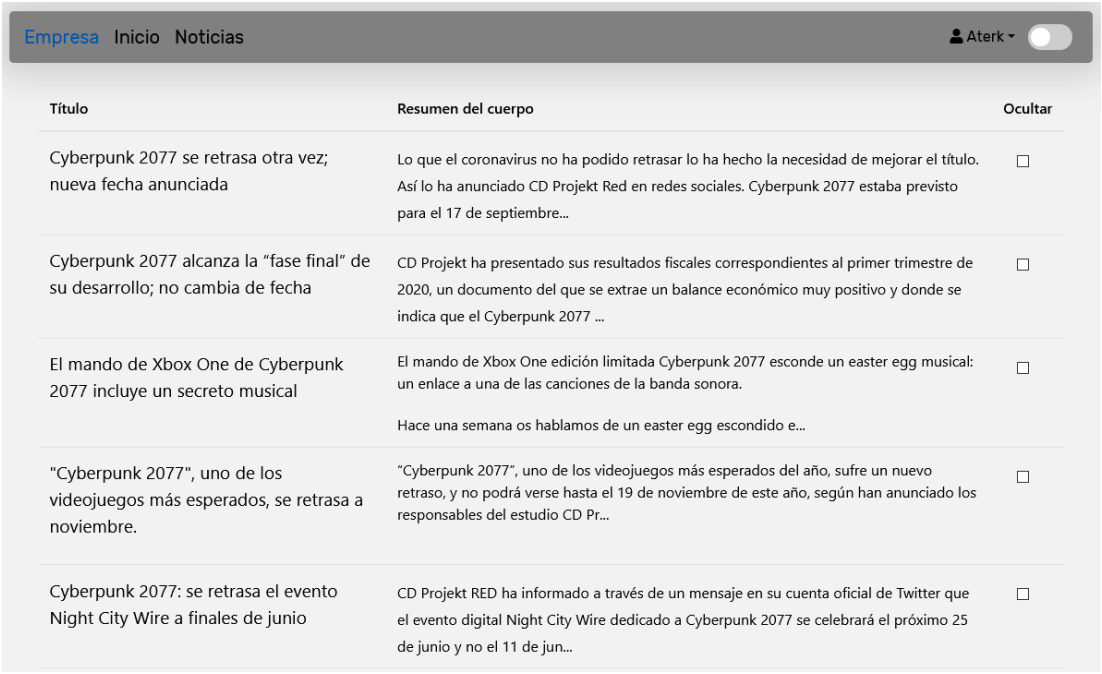

Figura A.12: Pantalla de editar noticias

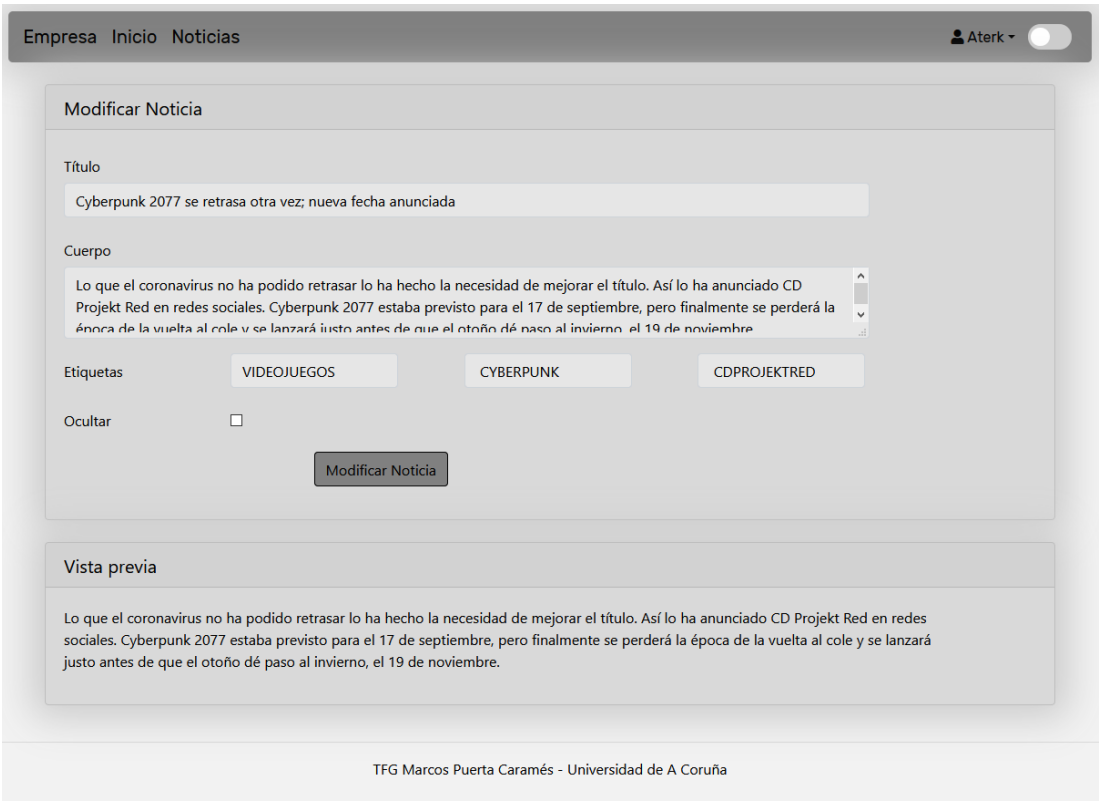

Figura A.13: Pantalla de editar una noticia

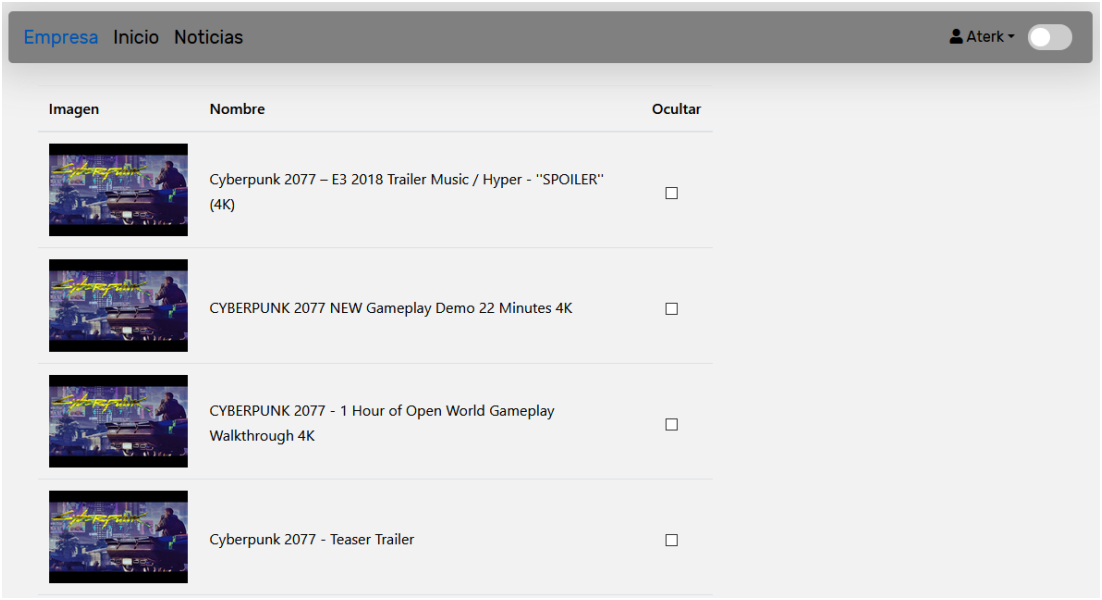

Figura A.14: Pantalla de editar vídeos

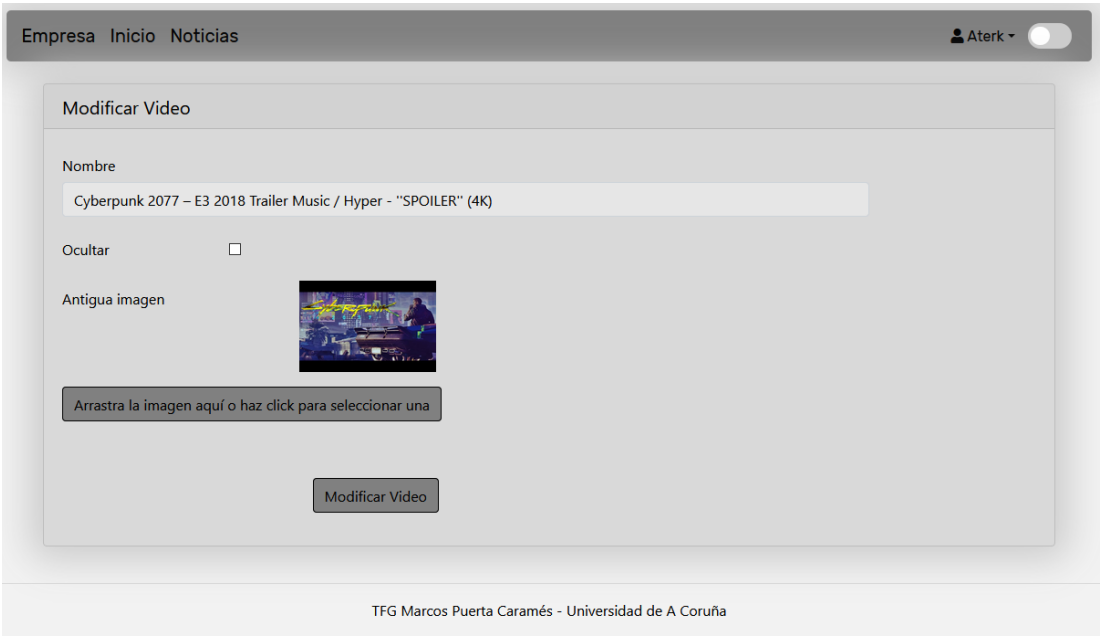

Figura A.15: Pantalla de editar un vídeo

A.1 Modo oscuro

También se ha implementado un modo oscuro, estas son pantallas representativas de como se visualiza.

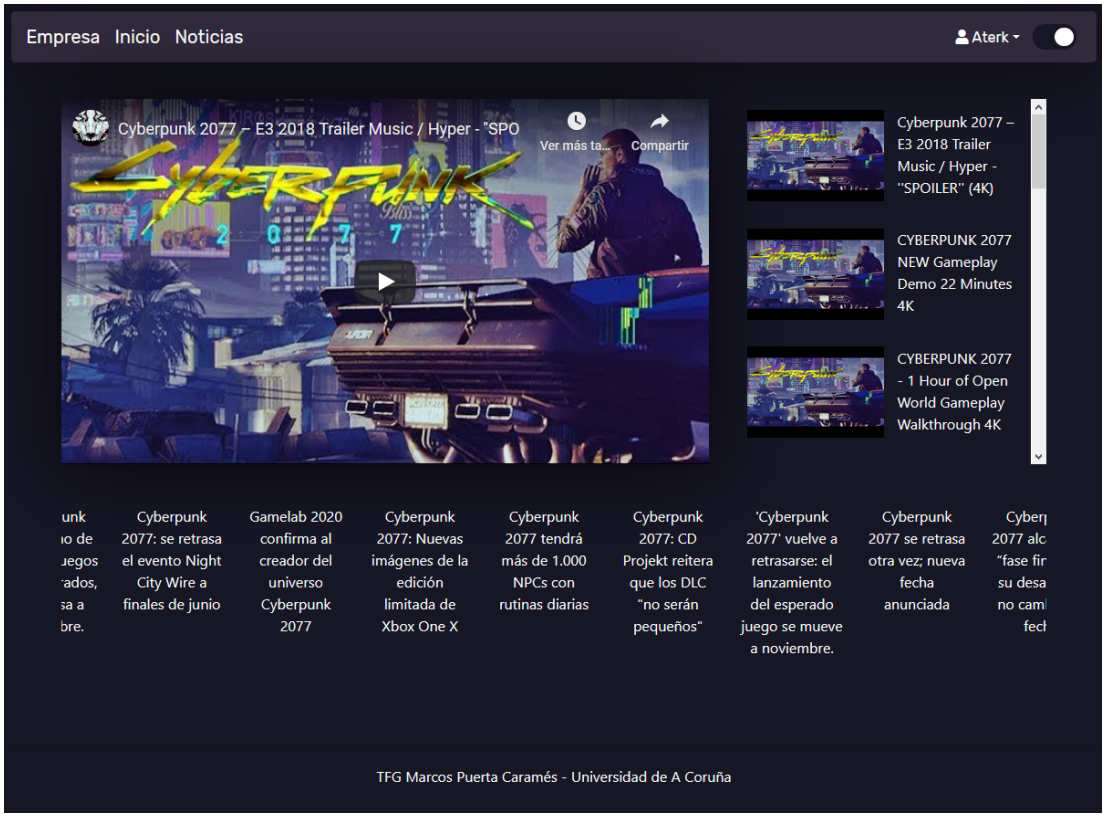

Figura A.16: Pantalla de Inicio modo noche

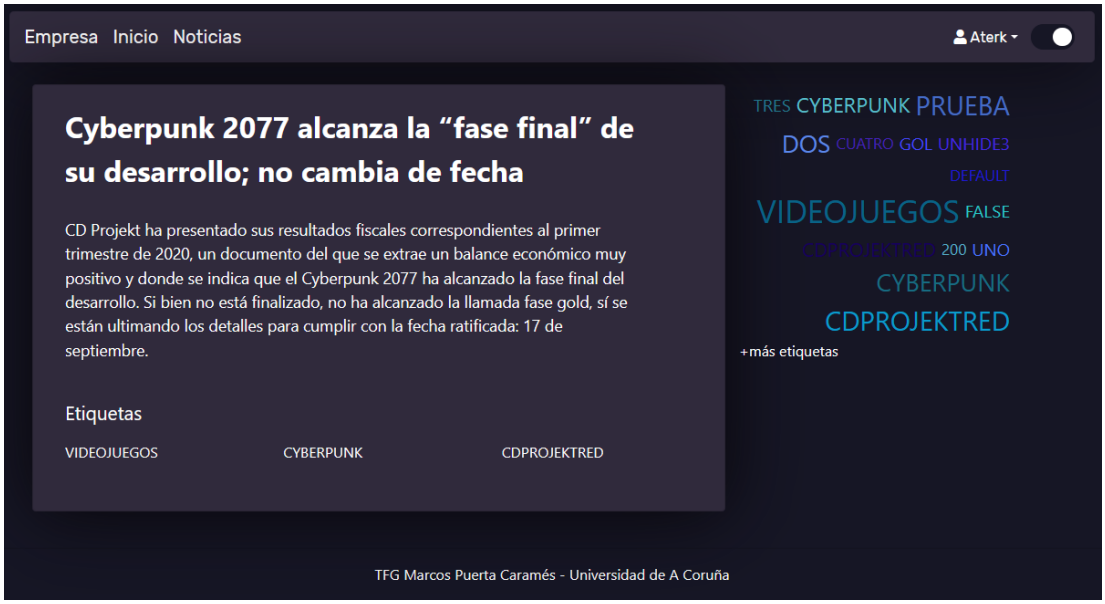

Figura A.17: Noticia modo oscuro

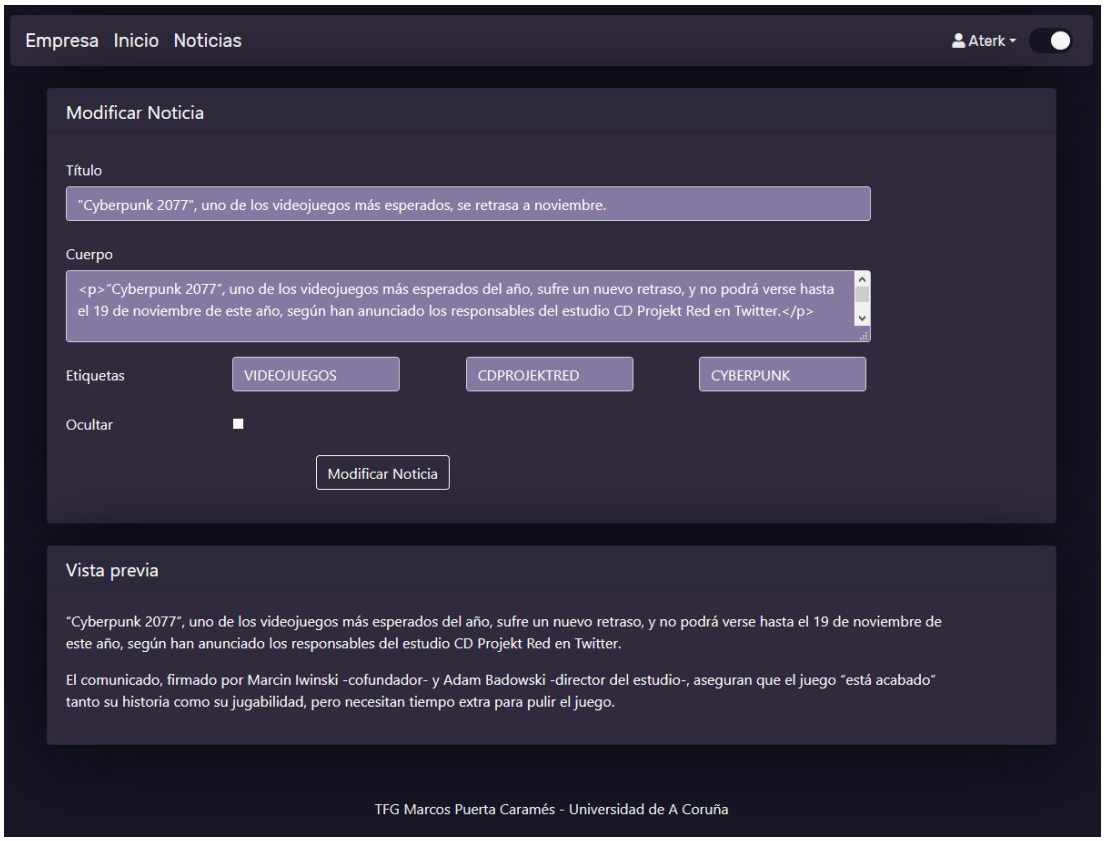

Figura A.18: Editar noticia modo oscuro

Lista de acrónimos

SQL *Structured Query Language.*

HTML *HyperText Markup Language.*

CSS *Cascading Style Sheets.*

POM *Project Object Model.*

REST *representational state transfer.*

JPA *Java Persistence API.*

HTTP *Hypertext Transfer Protocol.*

HTTPS *Hypertext Transfer Protocol Secure.*

MVC *Modelo-vista-controlador.*

DAO *Data Access Object.*

Java EE *Java Enterprise Edition.*

DTO *Data Transfer Object.*

Glosario

Front End es la parte de una aplicación que interactúa con los usuarios, es conocida como el lado del cliente.

Back End es todo con lo que el usuario se encuentra directamente en la web o aplicación.

Hosting es un servicio en línea que te permite publicar un sitio o aplicación web en Internet.

API es un conjunto de definiciones y protocolos que se utiliza para desarrollar e integrar el software de las aplicaciones.

Scrum es un proceso en el que se aplican de manera regular un conjunto de buenas prácticas para trabajar colaborativamente, en equipo, y obtener el mejor resultado posible de un proyecto. Estas prácticas se apoyan unas a otras y su selección tiene origen en un estudio de la manera de trabajar de equipos altamente productivos.

Framework es un conjunto estandarizado de conceptos, prácticas y criterios para enfocar un tipo de problemática particular que sirve como referencia, para enfrentar y resolver nuevos problemas de índole similar.

Bibliografía

[1] F. Morteo and N. Bocalandro, *Un enfoque práctico de SQL*. Ediciones Cooperativas, 2004.

[2] V. Rosselló Villán, "Qué son las metodologías ágiles y cuáles son sus ventajas empresariales," Apr 2020. [En línea]. Disponible en: https://www.iebschool.com/blog/que-son-metodologias-agiles-agile-scrum/

[3] M. Tena, "Metodología 'agile'. la revolución de las formas de trabajo," May 2020. [En línea]. Disponible en: https://www.bbva.com/es/metodologia-agile-la-revolucion-las-formas-trabajo/

[4] K. Lyytinen and D. R. Workshop, Eds., *Design requirements engineering: a ten-year perspective ; Design Requirements Workshop, Cleveland, OH, USA, June 3 - 6, 2007 ; revised and invited papers*, ser. Lecture notes in business information processing. Berlin: Springer, 2009, no. 14, meeting Name: Design Requirements Workshop OCLC: 845457200.

[5] M. Álvared Díaz, F. Bellas Permuy, and J. Raposo Santiago, "Diapositivas de la asignatura programación avanzada de la fic," 2019.

[6] J. M. Bishop, *C# 3.0 design patterns*, 1st ed. Beijing ; Sebastopol, CA: O'Reilly, 2008, oCLC: ocn145388447.

[7] S. de Informática. Universidad de Alicante, "Modelo vista controlador (mvc)." [En línea]. Disponible en: https://si.ua.es/es/documentacion/asp-net-mvc-3/1-dia/modelo-vista-controlador-mvc.html

[8] O. Blancarte, "Introducción a los patrones de diseño. un enfoque práctico." [En línea]. Disponible en: https://reactiveprogramming.io/blog/es/patrones-de-diseno/facade

[9] M. Fowler, "Data transfer object," 2010. [En línea]. Disponible en: https://martinfowler.com/eaaCatalog/dataTransferObject.html

[10] ——, “Data transfer object,” 2010. [En línea]. Disponible en: https://docs.microsoft.com/en-us/previous-versions/msp-n-p/ff649585(v=pandp.10)?redirectedfrom=MSDN